a-b

A5AC5024

lastlogin

The **lastlogin** command updates the file **/usr/adm/acct/sum/loginlog** to show the last date each user logged in. **runacct** normally calls this command.

monacct

The **monacct** command performs monthly (or periodic) accounting. **cron** should run this command once each month or accounting period. *number* indicates the month or period to process. The default *number* is the current month. This default is useful if **monacct** is run by **cron** on the first day of each month. The **monacct** command creates summary files in **/usr/adm/acct/fiscal** and restarts summary files in **/usr/adm/acct/sum**.

Daily reports are deleted (and thus inaccessible) each time **monacct** runs.

nulladm

The **nulladm** command creates *file*, assigns it permission code 664, and ensures that its owner and group are adm. (See "**chmod**" on page 160 for an explanation of file permissions.) Various accounting shell procedures call **nulladm**.

prctmp

The **prctmp** command displays the session record file created by the **acctcon1** command (normally **/usr/adm/acct/nite/ctmp**).

prdaily

The **prdaily** command formats a report of the day's accounting data. Use *mmdd* to specify a date other than the current day. The report resides in **/usr/adm/acct/sum/rprt***mmdd* where *mmdd* specifies the month and day of the report. **runacct** invokes this command to format a report of the previous day's accounting data.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-c	Reports exceptional resource usage by command, and may be used on the current day's accounting data only.
-l	Reports exceptional usage by login ID for the specified date.

prtacct

The **prtacct** command formats and displays any total accounting (**tacct**) file. You can specify a *heading* for the report by enclosing it in " " (double quotation marks).

Flags

-f*fieldspec*	Selects fields to be displayed, using the field selection mechanism of **acctmerg**.
-v	Produces verbose output in which more precise notation is used for floating-point numbers.

remove

The **remove** command deletes all **/usr/adm/acct/sum/wtmp***, **/usr/adm/acct/sum/pacct***, and **/usr/adm/acct/nite/lock*** files.

shutacct

The **shutacct** command turns process accounting off and adds a "reason" record to **/usr/adm/wtmp**. It is usually invoked during a system shutdown.

startup

The **startup** command turns on the accounting functions when the system is started up. It should be called by the **/etc/rc** command file.

turnacct

The **turnacct** command provides an interface to **accton** for turning process accounting **on** or **off**.

The **switch** flag turns accounting off, moves the current **/usr/adm/pacct** to the next free name in **/usr/adm/pacct***incr*, where *incr* is a number starting at 1 and increased by one for each additional **pacct** file. After moving the **pacct** file, **turnacct** turns accounting back on.

This command is usually called by **ckpacct**, which in turn is called by **cron**, keeping the **pacct** file down to a manageable size.

Files

/usr/adm/fee	Accumulator for fees charged to login names.
/usr/adm/pacct	Current file for process accounting.
/usr/adm/pacct*	Used if **pacct** gets large and during running of the daily accounting procedures.

/usr/adm/wtmp	Login/logout history file.
/usr/lib/acct/ptelus.awk	Shell procedure that calculates the limits for exceptional usage by login ID.
/usr/lib/acct/ptecms.awk	Shell procedure that calculates the limits of exceptional usage by command name.
/usr/adm/acct/nite	Working directory.
/usr/lib/acct	Holds all accounting commands.
/usr/adm/acct/sum	Summary directory.

Related Information

The following commands: "**acctcms**" on page 18, "**acctcom**" on page 20, "**acctcon**" on page 24, "**acctmerg**" on page 28, "**acctprc**" on page 30, "**chmod**" on page 160, "**cron**" on page 220, "**fwtmp**" on page 457, and "**runacct**" on page 848.

The **acct** system call and the **acct**, **utmp**, and **filesystems** files in *AIX Operating System Technical Reference.*

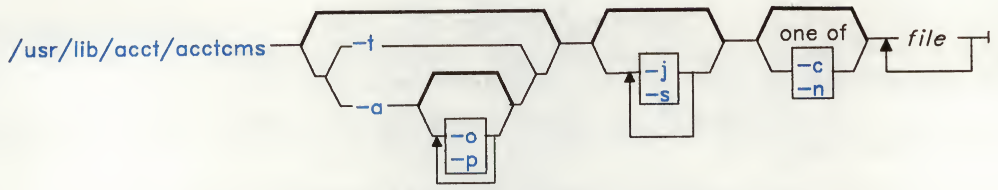
"Running System Accounting" in *Managing the AIX Operating System.*

acctcms

Purpose

Produces command usage summaries from accounting records.

Syntax



OL805421

Description

The **acctcms** command reads the specified *files*. It adds together all records for identically named processes, sorts them, and writes them to standard output in a binary format. Files are usually in the **acct** file format described in *AIX Operating System Technical Reference.*

When you use the **-o** and **-p** flags together, **acctcms** produces a report that combines prime- and nonprime-time. All the output summaries are of total usage except for number of times run, CPU minutes, and real minutes, which are split into prime and nonprime minutes.

A typical sequence for performing daily command accounting and for maintaining a running total is:

```
acctcms  file . . .   > today
cp  total  previoustotal
acctcms  -s  today  previoustotal  > total
acctcms  -a  -s  today
```

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-a	Displays output in ASCII summary format rather than binary summary format. Each output line contains the command name, the number of times the command was run, its total ***kcore-time***, its total ***CPU time***, its total ***real time***, its mean memory size (in K bytes), its mean CPU time per invocation of the command, and its ***CPU usage factor***. The listed times are all in minutes. **acctcms** normally sorts its output by total kcore-minutes. The unit kcore-minutes measures the amount of storage used (in K-bytes) multiplied by the amount of time it was in use.
-c	Sorts by total CPU time rather than total kcore-minutes.
-j	Combines under the heading `***other` all commands called only once.
-n	Sorts by the number of times the commands were called.
-o	Displays a command summary of nonprime-time commands only. You can use this flag with only the **-a** flag.
-p	Displays a command summary of prime-time commands only. You can use this flag with only the **-a** flag.
-s	Assumes that any named files that follow this flag are already in binary format.
-t	Processes all records as total accounting records. The default binary format splits each field into prime- and nonprime-time sections.

Related Information

The following commands: "**acct/***" on page 13, "**acctcom**" on page 20, "**acctcon**" on page 24, "**acctmerg**" on page 28, "**acctprc**" on page 30, "**fwtmp**" on page 457, and "**runacct**" on page 848.

The **acct** system call and the **acct** and **utmp** files in *AIX Operating System Technical Reference.*

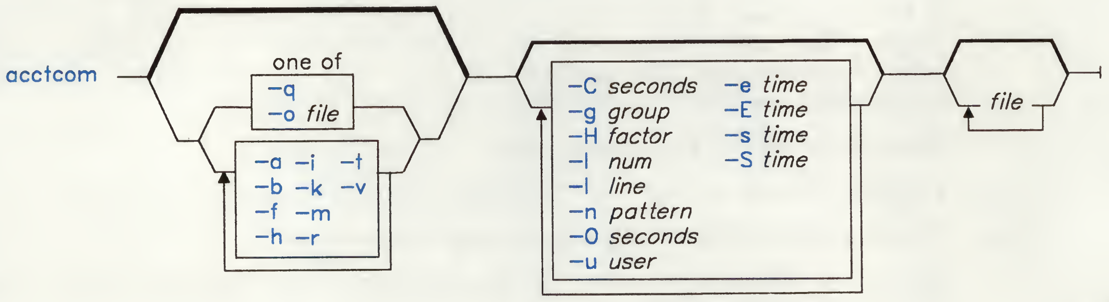
"Running System Accounting" in *Managing the AIX Operating System.*

acctcom

Purpose

Displays selected process accounting record summaries.

Syntax



OL805418

Description

The **acctcom** command reads from specified *files*, from standard input, or from **/usr/adm/pacct** and writes records (selected by flags) to standard output. The input file format is described under **acct** in *AIX Operating System Technical Reference.*

If you do not specify any *file* parameters and if standard input is assigned to a work station or to **/dev/null** (as it is when a process runs in the background), **acctcom** reads **/usr/adm/pacct** instead of standard input.

By default, if you specify any *file* parameters, **acctcom** reads each chronologically by process completion time. Usually, **/usr/adm/pacct** is the current file that you want **acctcom** to examine. Because the **ckpacct** procedure keeps this file from growing too large, a busy system may have several **pacct** files. All but the current file have the following path name:

```
/usr/adm/pacct?
```

where *?* (question mark) is an integer incremented each time a new file is created.

Each record represents one completed process. The default display consists of the command name, user name, tty name, start time, end time, real seconds, CPU seconds, and mean memory size (in kilobytes). These default items have the following headings in the output:

```
COMMAND                     START  END  REAL   CPU     MEAN
NAME     USER   TTYNAME TIME   TIME (SECS) (SECS)  SIZE(K)
```

By using the appropriate flags, you can also display the fork/exec flag (F), the system exit value (STAT), the ratio of total CPU time to elapsed time (HOG FACTOR), the product of memory used and elapsed time (KCORE MIN), the ratio of user time to total (system and user) time (CPU FACTOR), the number of characters transferred in input/output operations (CHARS TRNSFD), and the total number of blocks read or written (BLOCKS READ).

If a process ran with superuser authority, its name is prefixed with a # (pound sign). If a process is not assigned to a known work station (for example, when **cron** runs it), a ? (question mark) appears in the TTYNAME field.

Notes:

1. The **acctcom** command only reports on processes that have finished. Use the **ps** command to examine active processes.
2. If a specified time is later than the current time, it is interpreted as occurring on the previous day.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-a	Shows some average statistics about the processes selected. The statistics will be displayed after the output records.
-b	Reads backwards, showing the most recent commands first. This flag has no effect when **acctcom** reads standard input.
-C *seconds*	Shows only processes whose total CPU time (system time + user time), exceeds number of *seconds*.
-e *time*	Selects processes existing at or before the specified time. You can use the **NLTIME** environment variable to specify the order of hours, minutes, and seconds. The default order is *hh*[*mm*[*ss*]].
-E *time*	Selects processes ending at or before the specified time. You can use the **NLTIME** environment variable to specify the order of hours, minutes, and seconds. The default order is *hh*[*mm*[*ss*]]. If you specify the same time for

	both the **-E** and **-S** flags, **acctcom** displays the process that existed at the specified time.
-f	Displays the *fork/exec* flag and the system exit value columns in the output.
-g *group*	Selects processes belonging to *group*. You can specify either the group ID or the group name.
-h	Instead of mean memory size, shows the fraction of total available CPU time consumed by the process while it ran (***hog factor***). This factor is computed as: (total CPU time)/(elapsed time)
-H *factor*	Shows only processes that exceed *factor*. (See the **-h** flag for a discussion of how this factor is calculated.)
-i	Displays columns showing the number of characters transferred in read or write operations (the I/O counts).
-I *num*	Shows only processes transferring more than *num* characters.
-k	Instead of memory size, shows total kcore minutes.
-l *line*	Shows only processes belonging to work station **/dev/***line*.
-m	Shows mean main memory size. This flag is on by default. Specifying the **-h** or **-k** flags turns off **-m**.
-n *pattern*	Shows only commands matching *pattern*, where *pattern* is a regular expression like those in the **ed** command (see page 371), except that here you can use a + (plus sign) as a special symbol for one or more occurrences of the preceding character.
-o *file*	Copies selected process records to *file*, keeping the input data format. This flag suppresses writing to standard output.
-O *seconds*	Shows only processes with CPU system time exceeding *seconds*.
-q	Does not display any output records; just displays the average statistics that are displayed with the **-a** flag.
-r	Shows CPU factor. This factor is computed as: (user-time) / (system-time + user-time).
-s *time*	Shows only those processes that existed on or after the specified time. You can use the **NLTIME** environment variable to specify the order of hours, minutes, and seconds. The default order is *hh*[*mm*[*ss*]].
-S *time*	Shows only those processes starting at or after the specified time. You can use the **NLTIME** environment variable to specify the order of hours, minutes, and seconds. The default order is *hh*[*mm*[*ss*]].

-t	Shows separate system and user CPU times.
-u *user*	Shows only processes belonging to *user*. For *user*, you can give a user ID, a login name that is converted to a user ID, a # to select processes run with superuser authority, or a ? to select processes associated with unknown user IDs.
-v	Eliminates column headings from the output.

Files

/usr/adm/pacct	Current process accounting file.
/etc/passwd	User names and user IDs.
/etc/group	Group names and group IDs.

Related Information

The following commands: "**acctdisk, acctdusg**" on page 26, "**acctcms**" on page 18, "**acctcon**" on page 24, "**acctmerg**" on page 28, "**acctprc**" on page 30, "**acct/***" on page 13, "**fwtmp**" on page 457, "**ps**" on page 786, "**runacct**" on page 848, and "**su**" on page 1026.

The **acct** system call, the **acct** and **utmp** files and the **environment** miscellaneous facility in *AIX Operating System Technical Reference.*

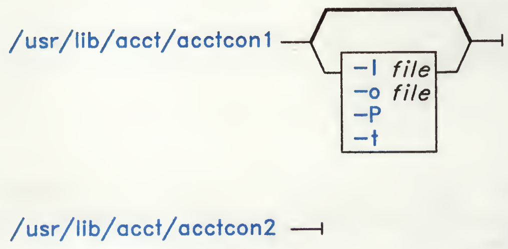
"Running System Accounting" and "Overview of International Character Support" in *Managing the AIX Operating System.*

acctcon

Purpose

Performs connect-time accounting.

Syntax



OL805233

Description

acctcon1

The **acctcon1** command converts a sequence of login and logoff records (read from standard input) to a sequence of login session records (written to standard output). its input should normally be redirected from **/usr/adm/wtmp**.

The **acctcon1** command displays, in ASCII format, the login device, user ID, login name, ***prime connect time*** (seconds), ***nonprime connect time*** (seconds), session starting time (numeric), and starting date and time (in date/time format). It also maintains a list of ports on which users are logged in. When it reaches the end of its input, it writes a session record for each port that still appears to be active. It normally assumes that its input is a current file, so that it uses the current time as the ending time for each session still in progress (see the **-t** flag on page 25).

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-l *file*	Writes to *file* a line-usage summary showing the line name, the number of minutes used, the percentage of total elapsed time used, the number of sessions charged, the number of logins, and the number of logoffs. This file helps track line usage and identify bad lines. All hang-ups, terminations of **login**, and terminations of the login shell cause the system to write logoff records, so the number of logoffs is often much higher than the number of sessions.
-o *file*	Writes to *file* an overall record for the accounting period, giving starting time, ending time, number of restarts, and number of date changes.
-p	Displays input only, showing line name, login name, and time in both numeric and date/time formats.
-t	Uses the last time found in the input as the ending time for any current processes instead of the current time. This is necessary in order to have reasonable and repeatable values for noncurrent files.

acctcon2

The **acctcon2** command converts a sequence of login session records, produced by the **acctcon1** command, into total accounting records.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Files

/usr/adm/wtmp	Login/logoff history file.

Related Information

The following commands: "**acctdisk, acctdusg**" on page 26, "**acctcms**" on page 18, "**acctcom**" on page 20, "**acctmerg**" on page 28, "**acctprc**" on page 30, "**acct/***" on page 13, "**fwtmp**" on page 457, "**init**" on page 521, "**login**" on page 584, and "**runacct**" on page 848.

The **acct** system call and the **acct** and **utmp** files in *AIX Operating System Technical Reference.*

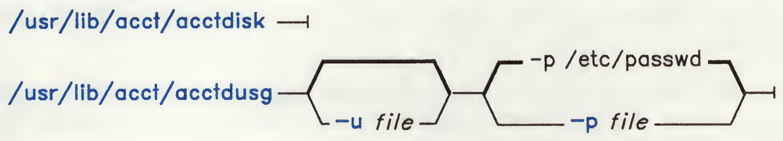
"Running System Accounting" in *Managing the AIX Operating System.*

acctdisk, acctdusg

Purpose

Performs disk-usage accounting.

Syntax



OL805192

Description

acctdisk

The **acctdisk** command reads lines from standard input that contain a user ID, the user's login name, and the number of disk blocks occupied by the user's files. It converts these lines to total accounting records that can be merged with other accounting records and writes those records to standard output.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

acctdusg

The **acctdusg** command reads a list of file names from standard input (usually piped from a **find / -print** command), computes disk resource usage (including ***indirect blocks***) using the login name of the owner of the files, and writes the results to standard output.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-p *file*	Searches *file* for login names and numbers, instead of searching **/etc/passwd**.
-u *file*	Places in *file* records of file names for which it does not charge.

Files

/etc/passwd	Used to convert login names to user IDs.
/usr/lib/acct	Directory holding all accounting commands.

Related Information

The following commands: "**acct/***" on page 13, "**acctcms**" on page 18, "**acctcom**" on page 20, "**acctcon**" on page 24, "**acctmerg**" on page 28, "**acctprc**" on page 30, "**fwtmp**" on page 457, and "**runacct**" on page 848.

The **acct** system call and the **acct** and **utmp** files in *AIX Operating System Technical Reference.*

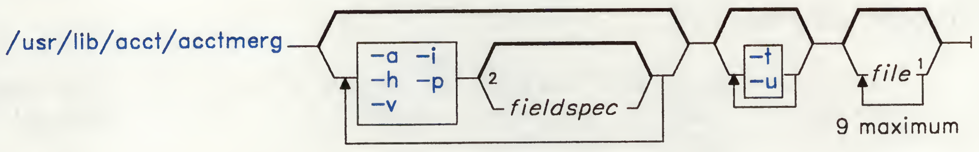
"Running System Accounting" in *Managing the AIX Operating System.*

acctmerg

Purpose

Merges total accounting files.

Syntax



[1]acctmerg always reads standard imput in addition to any named *files*.
[2]Do not put a blank between these items.

OL805234

Description

The **acctmerg** command reads records from standard input and up to nine additional *files*, all in the **tacct** binary format or the **tacct** ASCII format. It merges these by adding records with keys (normally user ID and name) that are identical, and expects the input records to be sorted by those key fields. It writes these merged records to standard output.

The optional *fieldspecs* allow you to select input or output fields. A field specification is a comma-separated list of fields or field ranges. Field numbers are in the order specified in the **tacct** file in *AIX Operating System Technical Reference*, with array sizes, except for the *ta_name* characters, taken into account. For example, `-h2-3,7,15-13,2` displays the login name, prime CPU and connect times, fee, queueing system, and disk usage data, and the login name again, in that order, with column headings. The default specification is "all fields" (`1-18` or `1-`), which produces very wide output lines containing all the available accounting data.

Queueing system, disk usage, or fee data can be converted into **tacct** records using the **-i***fieldspec* argument. For example, disk accounting records, produced by **acctdisk**, consist of lines containing the user ID, login name, number of blocks, and number of disk samples (always one). A file, **dacct**, containing these records can be merged into an existing total accounting file, **tacct**, with:

```
acctmerg  -i1-2,13,18  <dacct !. acctmerg  tacct  >output
```

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Flags

-a[*fieldspec*]	Produces output in the form of ASCII records.
-h[*fieldspec*]	Displays column headings. This flag implies **-a** but is effective with **-p** or **-v**.
-i[*fieldspec*]	Expects input files composed of ASCII records.
-p[*fieldspec*]	Displays input without processing.
-t	Produces a single record that contains the totals of all input.
-u	Summarizes by user ID rather than by user name.
-v[*fieldspec*]	Produces output in ASCII format, with more precise notation for floating-point numbers.

Example

The following sequence is useful for making repairs to any file in **tacct** format:

```
acctmerg  -v  <file1  >file2
     edit file2 as desired . . .
acctmerg  -a  <file2  >file1
```

Related Information

The following commands: "**acct/***" on page 13, "**acctcms**" on page 18, "**acctcom**" on page 20, "**acctcon**" on page 24, "**acctdisk, acctdusg**" on page 26, "**fwtmp**" on page 457, "**acctprc**" on page 30, and "**runacct**" on page 848.

The **acct** system call and the **acct** and **utmp** files in *AIX Operating System Technical Reference.*

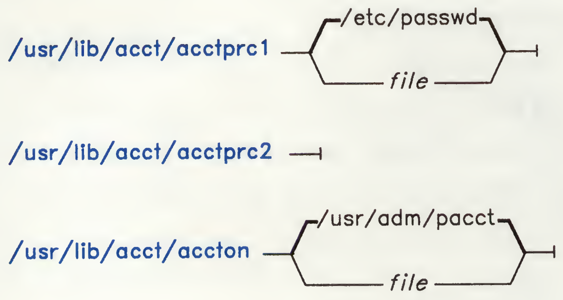
"Running System Accounting" in *Managing the AIX Operating System.*

acctprc

Purpose

Performs process accounting.

Syntax



OL805235

Description

acctprc1

The **acctprc1** command reads records from standard input that are in the **acct** format (described in *AIX Operating System Technical Reference*), adds the login names that correspond to user IDs, and then writes an ASCII record to standard output. This record contains the user ID, login name, prime CPU time, nonprime CPU time, the total number of characters transferred (in 512-byte units), the total number of blocks read and written, and mean memory size (in 64-byte units) for each process.

If specified, *file* contains a list of login sessions in **utmp** format (described in *AIX Operating System Technical Reference*), sorted by user ID and login name. By default, **acctprc1** gets login names from the password file, **/etc/passwd**. The information in *file* helps distinguish among different login names that share the same user ID.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

acctprc2

The **acctprc2** command reads (from standard input) the records written by **acctprc1**, summarizes them by user ID and name, and writes the sorted summaries to standard output as total accounting records.

Japanese Language Support Information

This command has not been modified to support Japanese characters.

accton

The **accton** command without arguments turns process accounting off. If you specify *file* (the name of an existing file), the kernel adds process accounting records to it (**/usr/adm/pacct** by default).

Japanese Language Support Information

This command has not been modified to support Japanese characters.

Files

/etc/passwd	Password file; contains user IDs.
/usr/adm/pacct	Contains process accounting records.

Related Information

The following commands: "**acct/***" on page 13, "**acctdisk, acctdusg**" on page 26, "**acctcms**" on page 18, "**acctcom**" on page 20, "**acctcon**" on page 24, "**acctmerg**" on page 28, "**fwtmp**" on page 457, and "**runacct**" on page 848.

The **acct** system call and the **acct** and **utmp** files in *AIX Operating System Technical Reference.*

"Running System Accounting" in *Managing the AIX Operating System.*

actman

Purpose

Permits interaction with multiple virtual terminals.

Syntax

```
actman —|
```

OL805323

Description

The **actman** command is the ***Activity Manager*** for the AIX Operating System. It is normally run by the AIX logger in the same manner as any program listed in the **/etc/passwd** file. Once started by the logger, **actman** creates the initial shell (**/bin/sh**) and monitors the number of open virtual terminals until all have been closed. It then exits to the AIX **init** process. If you try to end the initial shell when other virtual terminals are still open, **actman** restarts the initial shell.

To take advantage of the multiple virtual terminal capability, use the **open** command (see page 728) to execute another shell in a separate virtual terminal.

Notes:

1. You must log off of each existing shell to end your login session.
2. You do not need an Activity Manager if you do not have virtual terminal capabilities. Thus if you do not log in from the local console, **actman** overlays itself with the initial shell.

Related Information

The following command: "**open**" on page 728.

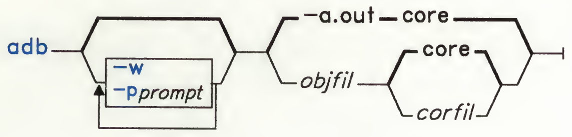
"Using Display Station Features" in *Using the AIX Operating System.*

adb

Purpose

Provides a general purpose debugger.

Syntax



OL805465

Description

The **adb** command provides a debugger for C and assembler language programs. With it, you can examine object and core files and provide a controlled environment for running a program.

Normally, *objfil* is an executable program file that contains a symbol table. If *objfil* does not contain a symbol table, the symbolic features of **adb** cannot be used, although the file can still be examined. The default *objfil* is **a.out**.

The *corfil* is assumed to be a core image file produced by running *objfil*. The default *corfil* is **core**.

While running, **adb** takes input from standard input and writes to standard output. **adb** does not recognize the **Quit** or **Interrupt** keys. These keys cause **adb** to wait for a new command.

In general, requests to **adb** are of the form

[*address*] [,*count*] [*command*] [;]

where *address* and *count* are expressions. The default *count* is 1. If *address* is specified, then the expression . (dot) is set to *address*.

The interpretation of an address depends on the context in which it is used. If a subprocess is being debugged, addresses are interpreted in the usual way in the address space of the subprocess. For more information, see "Addresses" on page 39.

You can enter more than one command at a time by separating the commands with a ; (semicolon).

Expressions

.	Specifies the last address used by a command; this is also known as the current address.
+	Increases the value of . (dot) by the current increment.
^	Decreases the value of . (dot) by the current increment.
"	Specifies the last address typed by a command.
integer	Specifies an octal number if *integer* begins with 0o, a hexadecimal number if preceded by 0x or #, or a decimal number if preceded by 0t; otherwise, a number interpreted in the current radix. The radix is initially 16.
'cccc'	Specifies the ASCII value of up to 4 characters. \ (slash) can be used to escape an ' (apostrophe).
< *name*	Reads the current value of *name*. *name* is either a variable name or a register name. **adb** maintains a number of variables (see "Variables" on page 39) named by single letters or digits. If *name* is a register name, the value of the register is obtained from the system header in *corfil*. The register names are **r0...r15, pc, ics, cs, mq**; the names **fp**, **pcp**, and **link** are recognized as synonyms for **r1**, **r14**, and **r15**.
symbol	Specifies a sequence of upper- or lower-case letters, underscores, or digits, not starting with a digit. The value of the *symbol* is taken from the symbol table in *objfil*. An initial _ (underscore) is prefixed to *symbol* if needed.
_symbol	Specifies, in C, the true name of an external symbol begins with _ (underscore), as does the name of the constant pool of an external function. It may be necessary to use this name to distinguish it from internal or hidden variables of a program.
.symbol	Specifies the entry point of the function named by *symbol*.
routine.name	Specifies the address of the variable *name* in the specified C routine. Both *routine* and *name* are *symbols*. If *name* is omitted, the value is the address of the most recently activated C stack frame corresponding to *routine*.
(*exp*)	Specifies the value of the expression *exp*.

Operators

Integers, symbols, variables, and register names can be combined with the following operators:

Unary

**exp*	Contents of location addressed by *exp* in *corefile.*
@*exp*	Contents of the location addressed by *exp* in *objfil.*
-*exp*	Integer negation.
~*exp*	Bitwise complement.

Binary

e1+e2	Integer addition.
e1-e2	Integer subtraction.
*e1*e2*	Integer multiplication.
e1%e2	Integer division.
e1&e2	Bitwise conjunction.
e1|e2	Bitwise disjunction.
e1#e2	*e1* rounded up to the next multiple of *e2.*

Binary operators are left associative and are less binding than unary operators.

Commands

You can display the contents of a text or data segment with the **?** (question mark) or the / (slash) command. The = (equal) command displays a given address in the specified format *f*. (The commands **?** and / may be followed by ***** (asterisk); see "Addresses" on page 39.)

?*f*	Displays, in the format *f*, the contents of the *objfil* starting at *address*. The value of **.** (dot) increases by the sum of the increment for each format letter.
/*f*	Displays, in the format *f*, the contents of the *corfil* starting at *address*. The value of **.** (dot) increases by the sum of the increment for each format letter.
=*f*	Displays the value of *address* in the format *f*. The **i** and **s** format letters are not meaningful for this command.

The format consists of one or more characters that specify print style. Each format character may be preceded by a decimal integer that is a repeat count for the format character. While stepping through a format, **.** (dot) increments by the amount given for each format letter. If no format is given, the last format is used.

The format letters available are as follows:

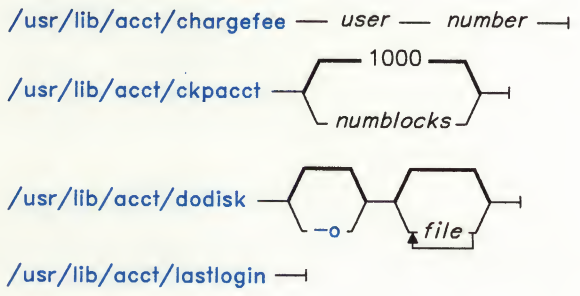
o 2	Prints 2 bytes in octal.
O 4	Prints 4 bytes in octal.
q 2	Prints 2 bytes in the current radix, unsigned.
Q 4	Prints 4 bytes in the current radix, unsigned.
d 2	Prints in decimal.
D 4	Prints long decimal.
x 2	Prints 2 bytes in hexadecimal.
X 4	Prints 4 bytes in hexadecimal.
u 2	Prints as an unsigned decimal number.
U 4	Prints long unsigned decimal.
b 1	Prints the addressed byte in the current radix, unsigned.
c 1	Prints the addressed character.
C 1	Prints the addressed character using the following escape conventions: 1. Prints control characters as ~ followed by the corresponding printing character. 2. Prints nonprintable characters as ~<*n*> where *n* is a hexadecimal value of the character. The character ~ prints as ~ ~.
s *n*	Prints the addressed character until a zero character is reached.
S *n*	Prints a string using the ~ escape convention. *n* specifies the length of the string including its zero terminator.
Y 4	Prints 4 bytes in date format (see "**ctime**" in *AIX Operating System Technical Reference*).
i n	Prints as instructions. *n* is the number of bytes occupied by the instruction.
a 0	Prints the value of **.** (dot) in symbolic form. Symbols are checked to ensure that they have an appropriate type as follows: / local or global data symbol ? local or global text symbol = local or global absolute symbol
p 4	Prints the addressed value in symbolic form using the same rules for symbol lookup as **a**.
t 0	When preceded by an integer, tabs to the next appropriate tab stop. For example, **8t** moves to the next 8-space tab stop.

acct/*

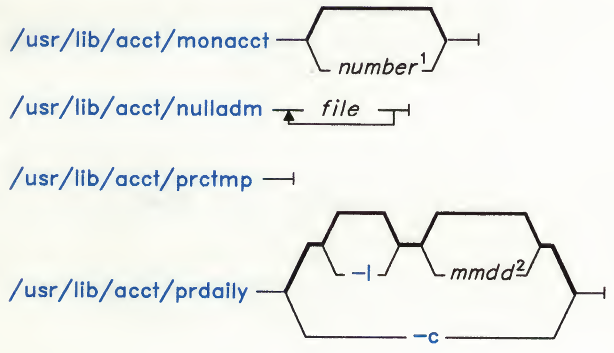
Purpose

Provides accounting shell procedures.

Syntax

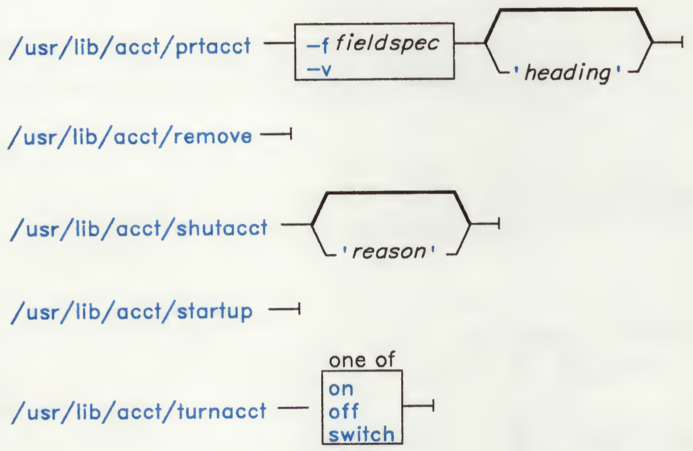


OL805236



[1] The default *number* is the current month.
[2] The default *mmdd* is the current day.

OL805237



OL805238

Description

Note: You should not share accounting files among nodes in a Distributed Services system. Each node should have its own copy of the various accounting files.

chargefee

The **chargefee** command charges the specified *number* of units to the specified *user*. *number* can have an integer or decimal value. It writes a record to **/usr/adm/fee**, to be merged with other accounting records by the **runacct** command.

ckpacct

The **ckpacct** command checks the size of **/usr/adm/pacct**. If the size exceeds the number specified in *numblocks*, **ckpacct** invokes **turnacct switch**. (The default value for *numblocks* is 1000.) If the number of free disk blocks in the **/usr** file system falls below 500, **ckpacct** automatically turns off the collection of process accounting records by invoking **turnacct off**. When 500 blocks are again available, accounting is activated again. This feature is sensitive to how frequently **ckpacct** is run (usually by **cron**).

dodisk

The **dodisk** command performs the disk-usage accounting functions. **cron** normally runs this command periodically. By default, it does disk accounting on the special files whose stanzas in **/etc/filesystems** contain the attribute **account=true**. If you specify the **-o** flag, it does a slower version of disk accounting by login directory.

The *file* parameter specifies the one or more file system names where disk accounting is to be done. If you specify any file names, disk accounting is done on only these file systems. If you do not specify **-o**, *file* names should be the special file names of mountable file systems. If you specify both **-o** and *file* names, the files should be mount points of mounted file systems.

r 0	Prints a space.
n 0	Prints a new line.
"..." 0	Prints the enclosed string.
^	Decreases **.** (dot) by the current increment. Nothing prints.
+	Increases **.** (dot) by 1. Nothing prints.
−	Decreases **.** (dot) decrements by 1. Nothing prints.

newline Repeats the previous command incremented with a *count* of 1.

[?/]**l***value mask* Words starting at **.** (dot) are masked with *mask* and compared with *value* until a match is found. If **L** is used, the match is for 4 bytes at a time instead of 2. If no match is found, **.** (dot) is unchanged; otherwise **.** (dot) is set to the matched location. If *mask* is omitted, -1 is used.

[?/]**w***value...* Writes the 2-byte *value* into the addressed location. If the command is **W**, write 4 bytes. If the command is **V**, write 1 byte. Alignment restrictions may apply when using **w** or **W**.

[?/]**m** *b1 e1 f1*[?/] Records new values for *b1, e1, f1.* If less than three expressions are given then the remaining map parameters are left unchanged. If the **?** or / is followed by ***** then the second segment (*b2, e2, f2*) of the mapping is changed. If the list is terminated by **?** or / then the file (*objfil* or *corfil* respectively) is used for subsequent requests. (For example, /**m?** causes / to refer to *objfil*).

>*name* Assigns . (dot) to the variable or register *name.*

! Calls a shell to read the rest of the line following **!**.

$*modifier* Miscellaneous commands. The available *modifiers* are:

<*file* Reads commands from *file* and returns to the standard input.

>*file* Sends output to *file.* If *file* is omitted, output returns to the standard output. *file* is created if it does not exist.

r Prints the general registers and the instruction addressed by **pc** and sets **.** (dot) to **pc**.

b Prints all breakpoints and their associated counts and commands.

c C stack back trace. If *address* is given, it is taken as the address of the current frame (instead of using the frame pointer register). If **C** is used, then the names and values of all automatic and static variables are printed for each active function. If *count* is given then only the first *count* frames are printed.

	e	Prints the names and values of external variables.
	w	Sets the output page width for *address*. The default is 80.
	s	Sets the limit for symbol matches to *address*. The default is 255.
	o	Sets the current radix to 8.
	d	Sets the current radix to *address* or 16, if none is specified.
	q	Exits **adb**.
	v	Prints all non-zero variables in octal.
	m	Prints the address map.
	p	Uses the remainder of the line as a prompt string.
:*modifier*	Manages a subprocess. Available *modifier*s are:	
	b*c*	Sets the breakpoint at *address*. The breakpoint runs *count* -1 times before causing a stop. Each time the breakpoint is encountered, the command *c* runs. If this command sets . (dot) to 0, the breakpoint causes a stop.
	d	Deletes the breakpoint at *address*.
	r	Runs *objfil* as a subprocess. If *address* is given explicitly, the program is entered at this point; otherwise, the program is entered at its standard entry point. *count* specifies how many breakpoints are to be ignored before stopping. Arguments to the subprocess may be supplied on the same line as the command. An argument starting with < or > causes the standard input or output to be established for the command. On entry to the subprocess, all signals are turned on.
	c*s*	Continues the subprocess with signal *s* (see the **signal** system call in *AIX Operating System Technical Reference*). If *address* is given, the subprocess is continued at this address. If no signal is specified, the signal that caused the subprocess to stop is sent. Breakpoint skipping is the same as for **r**.
	s*s*	Continues the subprocess in single steps *count* times. If there is no current subprocess, *objfil* is run as a subprocess. In this case no signal can be sent; the remainder of the line is treated as arguments to the subprocess.
	k	Stops the current subprocess, if one is running.

Variables

adb provides a number of variables. On entry to **adb**, the following variables are set from the system header in the *corfil*. If *corfil* does not appear to be a **core** file, then these values are set from *objfil*.

b	The base address of the data segment
d	The size of the data segment
e	The entry address of the program
m	The "magic" number (0405, 0407, 0410, or 0411)
s	The size of the stack segment
t	The size of the text segment.

Addresses

The address in a file associated with a written address is determined by a mapping associated with that file. Each mapping is represented by two triples (*b1, e1, f1*) and (*b2, e2, f2*). The *file address* that corresponds to a written *address* is calculated as follows:

$b1 \leq address < e1 => file\ address = address + f1 - b1$

or

$b2 \leq address < e2 => file\ address = address + f2 - b2$

Otherwise, the requested *address* is not legal. In some cases (for example, programs with separated I and D space) the two segments for a file may overlap. If a **?** or **/** is followed by an *****, then only the second triple is used.

The initial setting of both mappings is suitable for normal **a.out** and **core** files. If either file is not of the kind expected, then for that file *b1* is set to 0, *e1* is set to the maximum file size, and *f1* is set to 0; in this way, the whole file can be examined with no address translation.

In order for **adb** to be used on large files, all appropriate values are kept as signed 32-bit integers.

Flags

-p*prompt* Sets the prompt used by **adb** to *prompt*. If the prompt includes spaces, enclose the prompt in quotation marks.

-w Opens the *objfil* and *corfil* for writing. This flag makes either file if they do not exist.

Files

/dev/mem
/dev/swap
a.out
core

Related Information

The **ptrace** system call in *AIX Operating System Technical Reference.*

The **a.out** and **core** files in *AIX Operating System Technical Reference.*

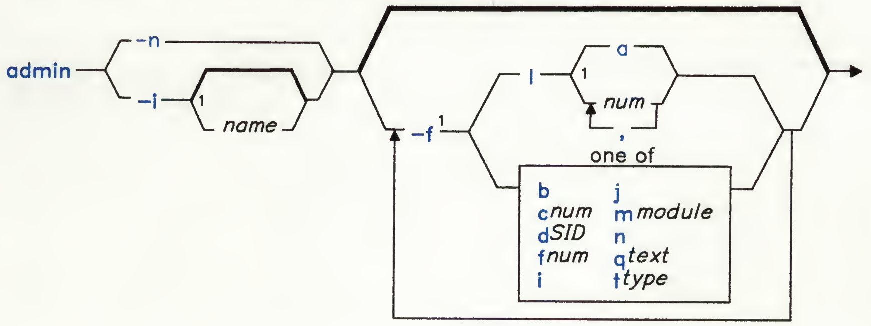
admin

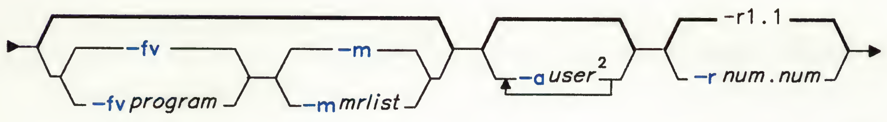
Purpose

Creates and initializes SCCS files.

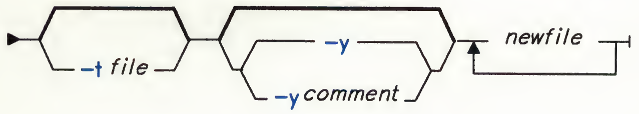
Syntax

To Create SCCS Files:





OL805376

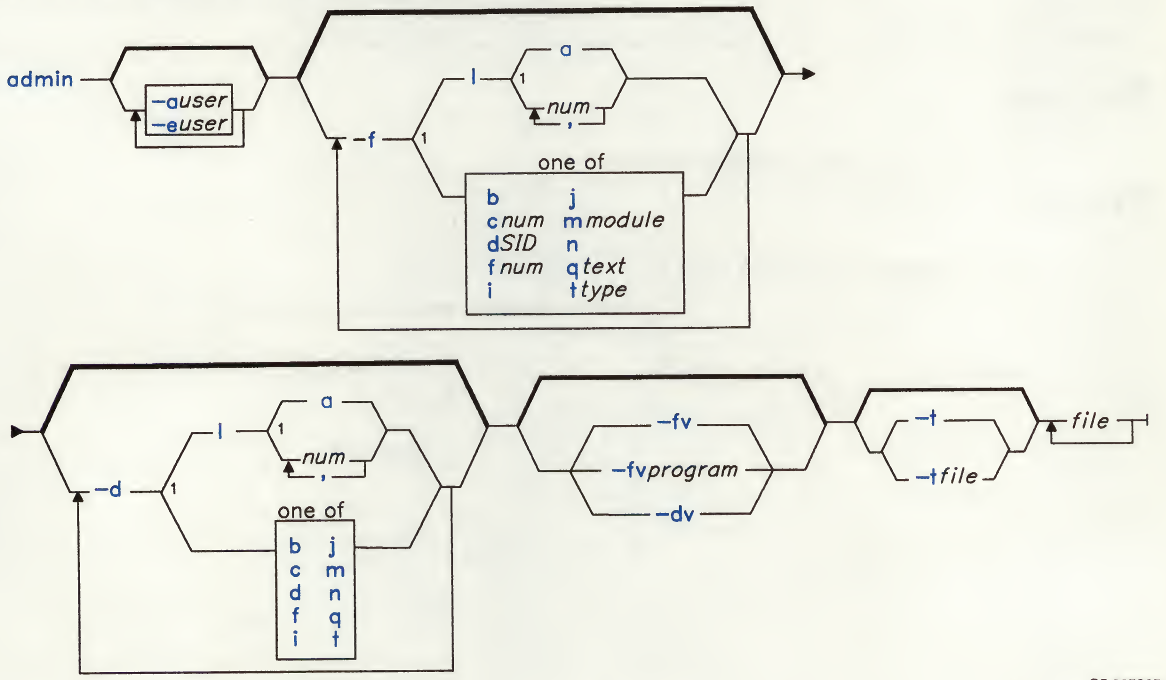


OL805160

2 If **-a** is never used to specify **users**, then any user can run **get -e** on the file.

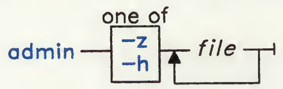
OL805417

To Change Existing SCCS Files:



OL805385

To Check and Correct Damaged SCCS Files:



OL805158

[1] Do not put a blank between these items.

OL805308

Description

The **admin** command creates new Source Code Control System (SCCS) *files* or changes specified parameters in existing SCCS *files*. These parameters control how the **get** command builds the files that you can edit. They also provide information about who can access the file, who can make changes, and when changes were made.

If the named *file* exists, **admin** modifies its parameters as specified by the flags. If it does not exist and you supply the **-i** or the **-n** flag, **admin** creates the new file and provides default values for unspecified flags. If you specify a directory name for *file*, **admin** performs the requested actions on all SCCS files in that directory (all files with the **s.** prefix). If you specify a **-** (minus) as a *file* name, **admin** reads standard input and interprets each line as the name of an SCCS file. An end-of-file character (**Ctrl-D**) ends input.

The **admin** command is most often used to create new SCCS files without setting parameters. See "Examples" on page 46 for the syntax used to create an SCCS file with no parameters set in the new file.

If you are not familiar with the delta numbering system, see *AIX Operating System Programming Tools and Interfaces* for more information.

SCCS File Conventions

All SCCS file names must have the form **s.***name*. New SCCS files are created with read-only permission. You must have write permission in the directory to create a file (see "**chmod**" on page 160 for an explanation of file permissions). **admin** writes to a temporary x-file, which it calls **x.***name*. The x-file has the same permissions as the original SCCS file if it already exists, and it is read-only if **admin** creates a new file. After successful completion of **admin**, the x-file is moved to the name of the SCCS file. This ensures that changes are made to the SCCS file only if **admin** does not detect any errors while it is running.

Directories containing SCCS files should be created with permission code 755 (read, write, and execute permissions for owner, read and execute permissions for group members and others). SCCS files themselves should be created as read-only files (444). With these permissions, only the owner can use non-SCCS commands to modify SCCS files. If a group can access and modify the SCCS files then the directories should include group write permission.

The **admin** command also uses a temporary lock file (called **z.***name*), to prevent simultaneous updates to the SCCS file by different users. See "SCCS Files" on page 478 for additional information on the **z.***name* file.

The following table contains the header flags that can be set with the **-f** flag and unset with the **-d** flags (see page 45). The header flags control the format of the g-file created with the **get** command (see "SCCS Files" on page 478 for details on the g-file).

Header Flag	Header Flag Purpose
b	Lets you use the **-b** flag of a **get** command to create branch deltas.
c*num*	Makes *num* the highest release number that a **get -e** can use. The value of *num* must be less than or equal to 9999. (Its default value is 9999.)
f*num*	Makes *num* the lowest release number that a **get -e** can retrieve. *num* must be greater than 0 and less than 9999. (Its default value is 1.)
d*SID*	Makes *SID* the default delta supplied to a **get** command.
i	Treats the `No id keywords (ge6)` message issued by the **get** or **delta** command as an error (see "Identification Keywords" on page 480).
j	Permits concurrent **get** commands for editing the same SID of an SCCS file. This allows multiple concurrent updates to the same version of the SCCS file.
l*num*[,*num*] . . .	Locks the releases specified by *num* . . . against editing, so that a **get -e** against one of these releases fails. You can lock all releases against editing by specifying **-fla** and unlock specific releases with the **-d** flag.
n	Causes **delta** to create a null delta in any releases that are skipped when a delta is made in a new release. For example, if you make delta 5.1 after delta 2.7, releases 3 and 4 will be null. The resulting null deltas can serve as points from which to build branch deltas. Without this flag, skipped releases do not appear in the the SCCS file.
q*text*	Substitutes *text* for all occurrences of the %Q% keyword in an SCCS text file retrieved by a **get** command. (See "Identification Keywords" on page 480 for more information on keywords.)
m*module*	Substitutes *module* for all occurrences of the %M% keyword in an SCCS text file retrieved by a **get** command. The default *module* is the name of the SCCS file without the **s.** prefix.
t*type*	Substitutes *type* for all %Y% keywords in a g-file retrieved by a **get**.
v[*program*]	Makes **delta** prompt for Modification Request (MR) numbers as the reason for creating a delta. *program* specifies the name of an MR number validity checking program (see "**delta**" on page 310). If **v** is set in the SCCS file, the **admin -m** flag must also be used, even if its value is null.

Figure 1. SCCS Header Flags

Flags

You can enter the flags and input file names in any order. All flags apply to all the files.

-a*user*	Adds the specified *user* to the list of users that can make sets of changes (***deltas***), to the SCCS file. *user* can be either a user name, a group name, or a group ID. Specifying a group name or number is the same as specifying the names of all users in that group. You can specify more than one **-a** flag on a single **admin** command line. If an SCCS file contains an empty user list, then anyone can add deltas. If a file has a user list, the creator of the file must be included in the list in order for the creator to make deltas to the file.
-d*hdrflag*	Removes the specified header flag from the SCCS file. You can specify this flag only with existing SCCS files. You can also specify more than one **-d** flag in a single **admin** command. See Figure 1 on page 44 for the header flags that **admin** recognizes.
-e*user*	Removes the specified *user* from the list of users allowed to make deltas to the SCCS file. Specifying a group ID is equivalent to specifying all *user* names common to that group. You can specify several **-e** flags on a single **admin** command line.
-f*hdrflag*[*value*]	Places the specified header flag and value in the SCCS file. You can specify more than one header flag in a single **admin** command. See Figure 1 on page 44 for the header flags that **admin** recognizes.
-h	Checks the structure of the SCCS file and compares a newly computed checksum with the checksum that is stored in the first line of the SCCS file. When the checksum value is not correct, the file has been improperly modified or has been damaged. This flag helps you detect damage caused by the improper use of non-SCCS commands to modify SCCS files, as well as accidental damage. The **-h** flag prevents writing to the file, so it cancels the effect of any other flags supplied. If an error message is returned indicating the file is damaged, use the **-z** flag to recompute the checksum. Then test to see if the file is corrected by using the **-h** flag again.
-i[*name*]	Gets the text for a new SCCS file from *name*. This text is the first delta of the file. If you specify the **-i** flag but you omit the file name, **admin** reads the text from standard input until it reaches end-of-file (**Ctrl-D**). If you do not specify the **-i** flag, but you do specify the **-n** flag, **admin** creates an empty SCCS file. **admin** can only create one file containing text at a time. If you are creating two or more SCCS files with one call to **admin**, you must use the **-n** flag, and the SCCS files created are empty.

-m[*mrlist*] Specifies a list of Modification Requests (MR) numbers to be inserted into the SCCS file as the reason for creating the initial delta. The **v** flag must be set. The MR numbers are validated if the **v** flag has a value (the name of an MR number validation program). **admin** reports an error if the **v** flag is not set or if MR validation fails.

-n Creates a new, empty SCCS file. Do not specify this flag when you use the **-i** flag.

-r*num.num* Inserts the initial delta into *num.num*, the release and version respectively. You can specify **-r** only if you also specify the **-i** or **-n** flag. If you do not specify this flag, the initial delta becomes Release 1, Version 1. Use this flag only when creating an SCCS file.

-t[*file*] Takes descriptive text for the SCCS file from *file*. If you use **-t** when creating a new SCCS file, you must supply a file name. In the case of existing SCCS files:

- Without a file name, **-t** causes removal of the descriptive text (if any) currently in the SCCS file.
- With a file name, **-t** causes text in the named file to replace the descriptive text (if any) currently in the SCCS file.

-y[*comment*] Inserts *comment* text into the initial delta in a manner identical to that of the **delta** command. Use this flag only when you create an SCCS file. If you do not specify a comment, **admin** inserts a line of the following form:

`date and time created` *YY/MM/DD HH:MM:SS* `by login`

-z Recomputes the SCCS file checksum and stores it in the first line of the SCCS file (see the **-h** flag on page 45).

Warning: Using **admin** with this flag on a damaged file can prevent future detection of the damage. This flag should only be used if the SCCS file is changed using non-SCCS commands because of a serious error.

Examples

1. To create an empty SCCS file named **s.prog.c**:

```
admin  -n  s.prog.c
```

2. To convert an existing text file into an SCCS file:

```
admin  -iprogram.c  s.prog.c
```

This converts the text file `program.c` into the SCCS file `s.prog.c`. The original file remains intact, but it is no longer needed. You must rename or delete it before you can use the **get** command on `s.prog.c`.

Related Information

The following commands: "**delta**" on page 310, "**ed**" on page 371, "**get**" on page 477, "**help**" on page 513, "**prs**" on page 781, and "**what**" on page 1213.

The **sccsfile** file in *AIX Operating System Technical Reference.*

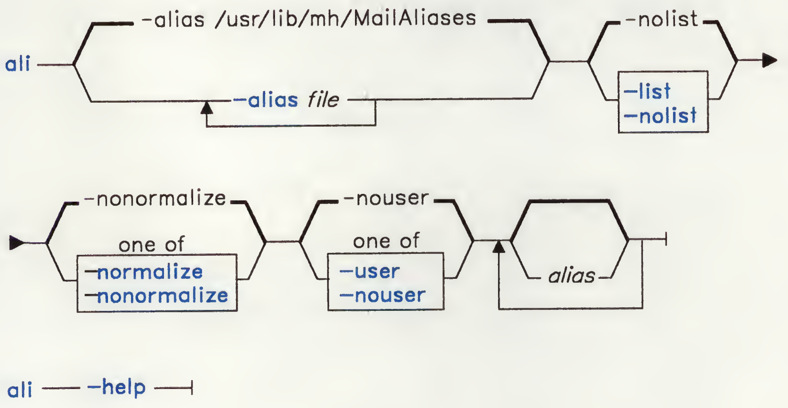
"Maintaining Different Versions of a Program" in *AIX Operating System Programming Tools and Interfaces.*

ali

Purpose

Lists mail aliases and their addresses.

Syntax



AJ2FL150

Description

The **ali** command is used to list mail aliases and the addresses that the aliases represent. **ali** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **ali** command searches the specified mail alias files for each given *alias*, and writes to standard output the addresses of each *alias*. If you specify the **-user** flag, **ali** interprets the **alias** arguments as actual addresses, searches the alias files for the addresses, and writes to standard output the aliases that contain definitions of the addresses. Thus, if you want to find the address of an alias, use the default **-nouser** flag. If you want to find the aliases that represent an address, use the **-user** flag.

Flags

-alias *file*	Specifies that *file* is a mail alias file to be searched for each given *alias*. The default alias file is **/usr/lib/mh/MailAliases**.
-help	Displays help information for the command.

-list	Displays each address on a separate line.
-nolist	Displays addresses separated by commas on as few lines as possible. This flag is the default.
-nonormalize	Does not attempt to convert local nicknames of hosts to their official host names. This flag is the default.
-normalize	Attempts to convert local nicknames of hosts to their official host names.
-nouser	Lists the addresses that the specified aliases represent. This flag is the default.
-user	Lists the aliases that contain the specified addresses. When the **-user** and **-nonormalize** flags are used together, the result may be a partial list of aliases that contain the specified addresses.

Files

/usr/lib/mh/MailAliases	The default mail alias file.
$HOME/.mh_profile	The MH user profile.
/etc/passwd	List of users.
/etc/group	List of groups.

Related Information

The following commands: "**comp**" on page 185, "**dist**" on page 336, "**forw**" on page 438, "**repl**" on page 821, "**send**" on page 893, "**whom**" on page 1222.

The **mh-alias** and **mh-profile** files in *AIX Operating System Technical Reference.*

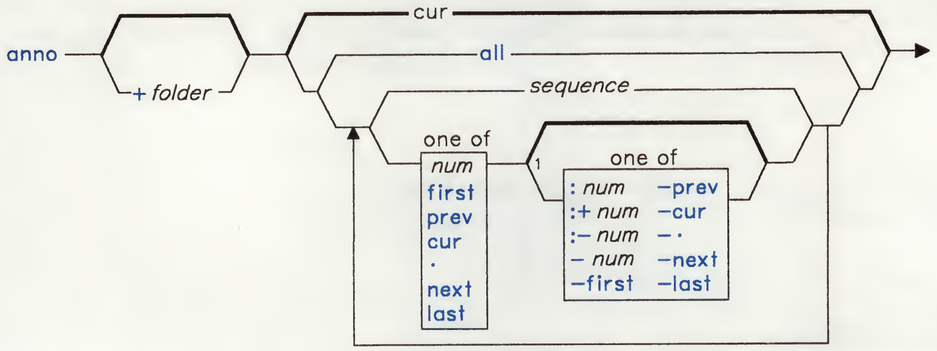
The "Overview of the Message Handling Package" in *Managing the AIX Operating System.*

anno

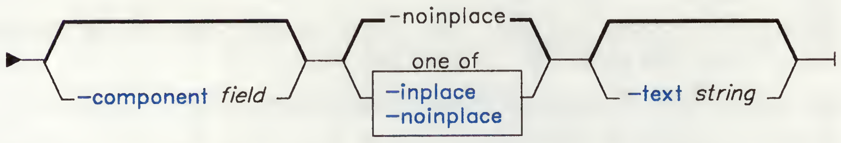
Purpose

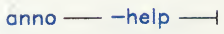
Annotates messages.

Syntax



AJ2FL221





AJ2FL166

[1] Do not put a blank between these items.

OL805308

Description

The **anno** command is used to annotate messages with specified text and dates. **anno** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **anno** command annotates messages with the lines:

```
field:date
field:body
```

Although **dist**, **forw**, and **repl** enable you to perform annotations, their annotations are limited to adding distribution information to messages. **anno** enables you to perform arbitrary annotations. The annotation fields must contain alphanumeric characters and dashes only.

Flags

-component *field*	Specifies the field name for the annotation text. The field name must be a valid message field name, consisting of alphanumeric characters and dashes only. If you do not specify this flag, **anno** prompts you for the name of the field.
+*folder msgs*	Specifies the messages that you want to annotate. *msgs* can be several messages, a range of messages, or a single message. You can use the following message references when specifying *msgs*:

num	**first**	**prev**
cur	**.**	**next**
last	**all**	*sequence*

The default message is the current message in the current folder. If several messages are specified, the first message annotated becomes the current message. If you specify a folder, that folder becomes the current folder.

-help	Displays help information for the command.
-inplace	Forces annotation to be done in place in order to preserve links to the annotated messages.
-noinplace	Does not perform annotation in place. This flag is the default.
-text *string*	Specifies the text to be annotated to the messages.

Profile Entries

Current-Folder:	Sets your default current folder.
Path:	Specifies your *user_mh_directory*.

Files

$HOME/.mh_profile	The MH user profile.

Related Information

The following commands: "**dist**" on page 336, "**forw**" on page 438, "**repl**" on page 821.

The **mh-profile** file in *AIX Operating System Technical Reference*.

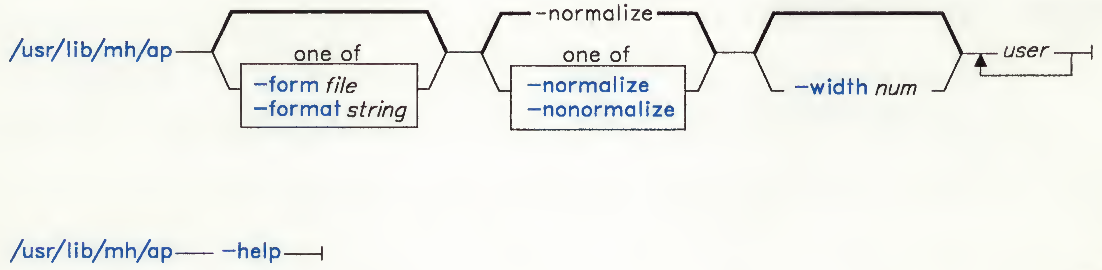
The "Overview of the Message Handling Package" in *Managing the AIX Operating System*.

ap

Purpose

Parses and reformats addresses.

Syntax



AJ2FL224

Description

The **ap** command is used to parse and reformat addresses. **ap** is not designed to be run directly by the user; it is designed to be called by other programs. The **ap** command is typically called by its full path name. The **ap** command is part of the MH (Message Handling) package.

The **ap** command parses each string specified as an address and attempts to reformat the string. The default output format for **ap** is the ARPA RFC822 standard. When the default format is used, **ap** displays an error message for each string it is unable to parse.

Flags

-form *file*	Reformats the given addresses into the alternate format described in *file*.
-format *string*	Reformats the given addresses into the alternate format specified by *string*. The default format string is: `%<{error}%{error}:%{address}%|%(putstr(proper{address}))%>`
-help	Displays help information for the command.
-nonormalize	Does not attempt to convert local nicknames of hosts to their official host names.

-normalize	Attempts to convert local nicknames of hosts to their official host names. This flag is the default.
-width *num*	Sets the maximum number of columns that **ap** uses to display dates and error messages. The default is the width of the display.

Files

$HOME/.mh_profile	The MH user profile.
/usr/lib/mh/mtstailor	The MH tailor file.

Related Information

Other MH commands: "**ali**" on page 48, "**dp**" on page 352, "**scan**" on page 871.

The **mh-alias**, **mh-format**, and **mh-profile** files in *AIX Operating System Technical Reference.*

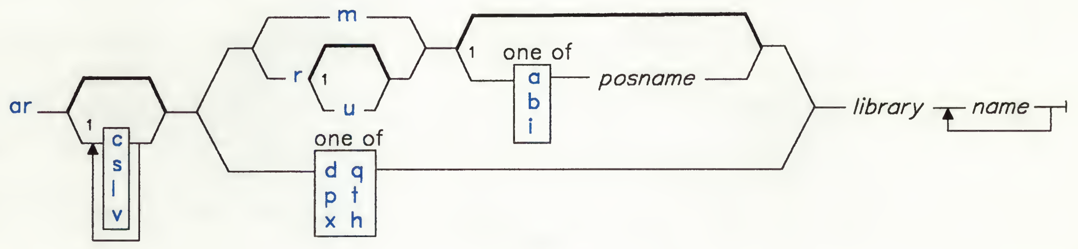
The "Overview of the Message Handling Package" in *Managing the AIX Operating System.*

ar

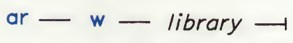
Purpose

Maintains portable libraries used by the linkage editor.

Syntax



OL805377



OL805349

[1] Do not put a blank between these items.

OL805308

Description

The **ar** command combines one or more named files into a single *library* file written in **ar** archive format. When **ar** creates a library, it creates headers in a transportable format; when it creates or updates a library, it rebuilds the symbol table that the ***linkage editor*** (the **ld** command) uses to make efficient multiple passes over object file libraries. See the **ar** file entry in *AIX Operating System Technical Reference* for information on the format and structure of portable archives and symbol tables.

Flags

In an **ar** command, you must list all selected flags together on the command line without blanks between them. You must specify one from the set **dhmpqrtxw**. You can also specify any number of optional flags from the set **abcilsuv**. If you select a positioning flag (**a**, **b**, or **i**), you must also specify the name of a file within *library* (*posname*), immediately following the flag list and separated from it by a blank.

a *posname*	Positions the named files after the existing file identified by *posname.*
b *posname*	Positions the named files before the existing file identified by *posname.*
c	Suppresses the normal message that is produced when *library* is created.
d	Deletes the named files from the library.
h	Sets the modification times in the member headers of the named files to the current date and time. If you do not specify any file names, **ar** sets the time stamps of all member headers.
i *posname*	Positions the named files before the existing file identified by *posname* (same as **b**).
l	Places temporary files in the current (local) directory instead of directory **/tmp**.
m	Moves the named files to some other position in the library. By default, it moves the named files to the end of the library. Use a positioning flag (**abi**) to specify some other position.
p	Writes to the standard output the contents of the named *file*s or all files in a *library* if you do not specify any files.
q	Adds the named files to the end of the library. Positioning flags, if present, do not have any effect. Note that this process does not check to see if the named files are already in the library. In addition, if you name the same file twice, it may be put in the library twice.
r	Replaces a named file if it already appears in the library. Since the named files occupy the same position in the library as the files they replace, a positioning flag does not have any additional effect. When used with the **u** flag (update), **r** replaces only files modified since they were last added to the library file. If a named file does not already appear in the library, **ar** adds it. In this case, positioning flags do affect placement. If you do not specify a position, new files are placed at the end of the library. If you name the same file twice, it may be put in the library twice.
s	Forces the regeneration of the library symbol table whether or not **ar** modifies the library contents. Use this flag to restore the library symbol table after using the **strip** command on the library.
t	Writes to the standard output a table of contents for the library. If you specify file names, only those files appear. If you do not specify any files, **t** lists all files in the library.
u	Copies only files which have been changed since they were last copied (see the **r** flag discussed previously).

v Writes to standard output a verbose file-by-file description of the making of the new library. When used with the **t** flag, it gives a long listing similar to that of the **ls -l** command, described under "**ls**" on page 595. When used with the **x** flag, it precedes each file with a name. When used with the **h** flag, it lists the member name and the updated modification times.

The environment variables **NLLDATE** and **NLTIME** control the format of the archive date and time.

w Displays the archive symbol table. Each symbol is listed with the name of the file in which the symbol is defined.

x Extracts the named files by copying them into the current directory. These copies have the same name as the original files, which remain in the library. If you do not specify any files, **x** copies all files out of the library. This process does not alter the library.

Examples

1. To create a library:

   ```
   ar  vq  lib.a  strlen.o  strcpy.o
   ```

   If `lib.a` does not exist, then this creates it and enters into it copies of the files `strlen.o` and `strcpy.o`. If `lib.a` does exist, then this adds the new members to the end without checking for duplicate members. The **v** flag sets verbose mode, in which **ar** displays progress reports as it proceeds.

2. To list the table of contents of a library:

   ```
   ar  vt  lib.a
   ```

   This lists the table of contents of `lib.a`, displaying a long listing similar to **ls -l**. To list only the member file names, omit the **v** flag.

3. To replace or add new members to a library:

   ```
   ar  vr  lib.a  strlen.o  strcat.o
   ```

   This replaces the members `strlen.o` and `strcat.o`. If `lib.a` was created as shown in Example 1, then the `strlen.o` member is replaced. A member named `strcat.o` does not already exist, so it is added to the end of the library.

4. To specify where to insert a new member:

   ```
   ar vrb strlen.o lib.a strcmp.o
   ```

   This adds `strcmp.o`, placing the new member before `strlen.o`.

5. To update a member if it has been changed:

   ```
   ar  vru  lib.a  strcpy.o
   ```

This replaces the existing `strcpy.o` member, but only if the file `strcpy.o` has been modified since it was last added to the library.

6. To change the order of the library members:

   ```
   ar vma strcmp.o lib.a strcat.o strcpy.o
   ```

   This moves the members `strcat.o` and `strcpy.o` to positions immediately after `strcmp.o`. The relative order of `strcat.o` and `strcpy.o` is preserved. In other words, if `strcpy.o` preceded `strcat.o` before the move, then it still does.

7. To extract library members:

   ```
   ar vx lib.a strcat.o strcpy.o
   ```

   This copies the members `strcat.o` and `strcpy.o` into individual files named `strcat.o` and `strcpy.o`, respectively.

8. To extract and rename a member:

   ```
   ar p lib.a strcpy.o >stringcopy.o
   ```

   This copies the member `strcpy.o` to a file named `stringcopy.o`.

9. To delete a member:

   ```
   ar vd lib.a strlen.o
   ```

   This deletes the member `strlen.o` from the library `lib.a`.

Files

/tmp/ar*	Temporary files.

Related Information

The following commands: "**backup**" on page 88, "**ld**" on page 557, "**lorder**" on page 591, "**make**" on page 625, "**nm**" on page 705, "**size**" on page 949, and "**strip**" on page 1017.

The **a.out** and **ar** files and **environment** miscellaneous facility in *AIX Operating System Technical Reference.*

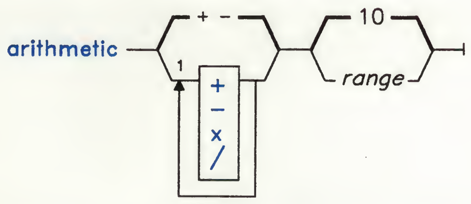
The "Overview of International Character Support" in *Managing the AIX Operating System.*

arithmetic

Purpose

Tests arithmetic skills.

Syntax



¹ Do not put a blank between these items.

OL805164

OL805308

Description

The **arithmetic** command displays simple arithmetic problems and waits for you to enter an answer. If your answer is correct, the program displays `Right!` and presents a new problem. If your answer is wrong, it displays `What?` and waits for another answer. Every 20 problems, **arithmetic** displays the number of correct and incorrect responses and the time required to answer.

The **arithmetic** command does not give the correct answers to the problems it displays. It provides practice rather than instruction in performing arithmetic calculations.

The *range* is a decimal number specifying the permissible range of all numbers (except answers). The default range is 10. At the start, all numbers within this range are equally likely to appear. If you make a mistake, the numbers in the problem you missed become more likely to reappear.

To quit the game, press **INTERRUPT (Alt-Pause)**; **arithmetic** displays the final game statistics and exits.

Flags

Two types of optional flags modify the action of **arithmetic**. The first set specifies the type of arithmetic problem:

+	Specifies addition problems.
-	Specifies subtraction problems.
x	Specifies multiplication problems.
/	Specifies division problems.

If you do not select any flags, **arithmetic** selects addition and subtraction problems. If you give more than one problem specifier (+**-x**/), the program mixes the specified types of problems in random order.

Examples

1. To drill on addition and subtraction of integers from 0 to 10:

```
/usr/games/arithmetic
```

2. To drill on addition, multiplication, and division of integers from 0 to 50:

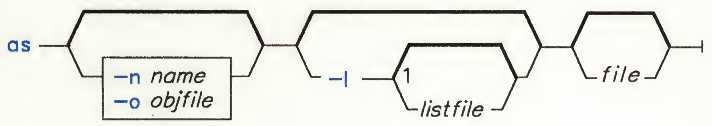
```
/usr/games/arithmetic  +x/  50
```

as

Purpose

Assembles a source file.

Syntax



OL805165

^{1} Do not put a blank between these items.

OL805308

Description

The **as** command reads and assembles the named *file* (conventionally this file ends with a **.s** suffix). If you do not specify a *file*, **as** reads and assembles standard input. It stores its output, by default, in a file named **a.out**. The output file is executable if no errors occur and if there are no unresolved external references.

Flags

-l[*listfile*]	Produces an assembler listing. If you do not specify a file name, a default name is produced by replacing the **.s** extension of the source file name with an **.lst** extension.
-n *name*	Specifies the name that appears in the header of the assembler listing. By default, the header contains the name of the assembler source file.
-o *objfile*	Writes the output of the assembly process to the specified file instead of to **a.out**.

Files

a.out	Default output file.

Related Information

The following commands: "**cc**" on page 140 and "**ld**" on page 557.

The **a.out** file in *AIX Operating System Technical Reference.*

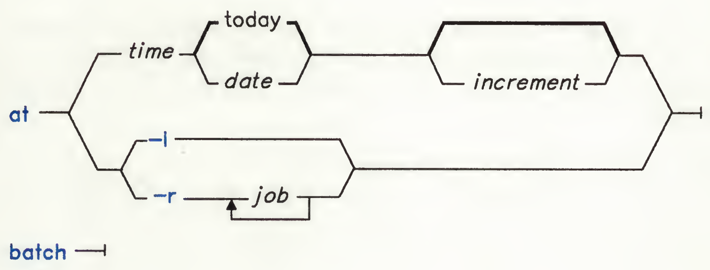
The discussion of **as** in *Assembler Language Reference* and *AIX Operating System Programming Tools and Interfaces.*

at, batch

Purpose

Runs commands at a later time.

Syntax



OL805002

Description

The **at** and **batch** commands read from standard input the names of commands to be run at a later time:

- **at** allows you to specify when the commands should be run.
- **batch** runs jobs when the system load level permits.

Both **at** and **batch** mail you all output from standard output and standard error for the scheduled commands, unless you redirect that output. They also write the job number and the scheduled time to standard error.

Variables in the shell environment, the current directory, **umask**, and **ulimit** are retained when the commands are run. Open file descriptors, traps, and priority are lost.

You can use **at** if your name appears in the file **/usr/lib/cron/at.allow**. If that file does not exist, **at** checks the file **/usr/lib/cron/at.deny** to determine if you should be denied access to **at**. If neither file exists, only the superuser can submit a job. The **allow/deny** files contain one user name per line. If **at.allow** does exist, the superuser's login name must be included in it for the superuser to be able to use the command.

The required *time* parameter can be one of the following:

1. A number followed by an optional suffix. **at** interprets one- and two-digit numbers as hours. It interprets four digits as hours and minutes. The **NLTIME** environment variable specifies the order of hours and minutes. The default order is the hour followed by the minute. You can also separate hours and minutes with a : (colon). The default order is *hour:minute.*

   In addition, you may specify a suffix of **am**, **pm**, or **zulu**. If you do not specify **am** or **pm**, **at** uses a 24 hour clock. The suffix **zulu** indicates that the time is **GMT** (Greenwich Mean Time). The **NLTMISC** environment variable controls the suffixes that **at** recognizes.

2. **at** also recognizes the following keywords as special *time*s: **noon**, **midnight**, and **now**. Note that you can use the special word **now** only if you also specify a *date* or an *increment.* Otherwise, **at** tells you: `too late`. The **NLTSTRS** environment variable controls the additional keywords that **at** recognizes.

You may specify the *date* parameter as either a month name and a day number (and possibly a year number preceded by a comma), or a day of the week. The **NLDATE** environment variable specifies the order of the month name and day number (by default, month followed by day). The **NLLDAY** environment variable specifies long day names; **NLSDAY** and **NLSMONTH** specify short day and month names. (By default, the long name is fully spelled out; the short name abbreviated to three characters.) **at** recognizes two special "days," **today** and **tomorrow** by default. (The **NLTSTRS** environment variable specifies these special days.) **today** is the default *date* if the specified time is later than the current hour; **tomorrow** is the default if the time is earlier than the current hour. If the specified month is less than the current month (and a year is not given), next year is the default year. The optional *increment* can be one of the following:

1. A + (plus sign) followed by a number and one of the following words: **minute**[s], **hour**[s], **day**[s], **week**[s], **month**[s], **year**[s] (or their non-English equivalents).
2. The special word **next** followed by one of the following words: **minute**[s], **hour**[s], **day**[s], **week**[s], **month**[s], **year**[s] (or their non-English equivalents).

The **NLTUNITS** environment variable specifies the non-English equivalents of the English defaults.

Flags

-l	Reports your scheduled jobs.
-r *job* . . .	Removes *job*s previously scheduled by **at** or **batch**, where *job* is the number assigned by **at** or **batch**. If you do not have superuser authority (see "**su**" on page 1026), you can remove only your own jobs.

Examples

1. To schedule the command from the terminal, use a command similar to one of the following:

   ```
   at  5 pm  Friday uuclean
   ```
   Ctrl-D
   ```
   at  now  next  week uuclean
   ```
   Ctrl-D
   ```
   at  now  +  2  days uuclean
   ```
   Ctrl-D

2. To run **uuclean** at 3:00 in the afternoon on the 24th of January, use any one of the following commands:

   ```
   echo  uuclean  |  at  3:00  pm  January  24
   echo  uuclean  |  at  3pm  Jan  24
   echo  uuclean  |  at  1500  jan  24
   ```

3. To run a job when the system load permits:

   ```
   batch  <<!
   longjob  2>&1  >outfile  |  mail  myID
   !
   ```

   This example shows the use of a ***here document*** to send standard input to **at** (see "Inline Input Documents" on page 928).

   The order of redirections is important here, so that only error messages are sent into the pipe to the **mail** command. If you reverse the order, both standard error and standard output are sent to `outfile` (see the discussion of "Input and Output Redirection Using File Descriptors" on page 928 for details).

4. To have a job reschedule itself, invoke **at** from within the shell procedure by including code similar to the following within the shell file:

   ```
   echo  "sh  shellfile"  |  at  now  tomorrow
   ```

5. To list the jobs you have sent to be run later:

   ```
   at  -l
   ```

6. To cancel jobs:

   ```
   at  -r  103  227
   ```

   This cancels jobs 103 and 227. Use **at -l** to list the job numbers assigned to your jobs.

Files

/usr/lib/cron	Main cron directory.
/usr/lib/cron/at.allow	List of allowed users.
/usr/lib/cron/at.deny	List of denied users.
/usr/spool/cron/atjobs	Spool area.

Related Information

The following commands: "**cron**" on page 220, "**kill**" on page 552, "**mail, Mail**" on page 608, "**nice**" on page 699, "**ps**" on page 786, and "**sh**" on page 913.

The **environment** special facility in *AIX Operating System Technical Reference.*

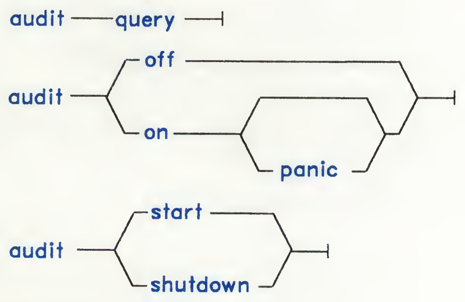
"Running Commands at Pre-set Times" and "Overview of International Character Support" in *IBM RT Managing the AIX Operating System.*

audit

Purpose

Controls system auditing

Syntax



AJ2FL131

Description

The **audit** command controls system auditing. The **audit** command enables or disables auditing, and no audit records are generated if the audit system is disabled. You must have superuser authority to run this command.

The following arguments are available with the **audit** command:

query Gives the current status of the auditing system in the form:

```
auditing on            (or auditing off)
bin processing off     (or bin manager is process number)
audit events:
                audit class:  auditevent,auditevent,auditevent
                (or none)
```

start Sets up and enables the auditing system. The system initialization file, **/etc/rc**, normally includes the **audit start** command.

shutdown Terminates the operation of the auditing system. This argument forces all audit records out to the audit trail. It then empties the bin files, invalidates the current configuration, and stops the collection process until the next **audit start** command is given.

off	Stops the auditing system, but leaves the auditing collection configuration valid; no records are lost, and the collection process pauses temporarily until the **audit on** or **audit shutdown** command is given.
on [panic]	Enables auditing. Audit records are generated for enabled events. This argument assumes that Bin audit collection has already been established (see the discussion of information collection in *Managing the AIX Operating System*). **Note:** If you specify the **panic** option, reliable long-term storage of audit records is required. The **auditbin** procedure must already have been started to manage the disposition of audit bins. If the kernel is unable to write a record into a bin for archival, the audit system shuts down the system.

To start the auditing system, **audit** reads configuration information from the **/etc/security/config** file. To start auditing, **audit** does the following:

1. Starts the **auditbin** collection procedure if Bin audit collection is enabled. The procedure synchronously recovers any unprocessed bins.
2. Enables the audit classes defined in the **auditclasses** stanza of the **/etc/security/config** file.
3. Starts auditing.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Files

/etc/security/audit/events	Lists audit events.
/etc/security/config	Specifies the audit configuration.
/etc/security/audit/cmds	Lists audit bin backend programs.
/etc/security/passwd	Lists audit classes for which users will be audited.

Related Information

The following commands: "**auditbin**" on page 71 and "**auditpr**" on page 73.

The **audit**, **auditbin**, **auditevents**, **auditlog**, and **auditproc** system calls in the *AIX Operating System Technical Reference.*

The **audit**, **events**, **passwd**, and **config** file formats in the *AIX Operating System Technical Reference.*

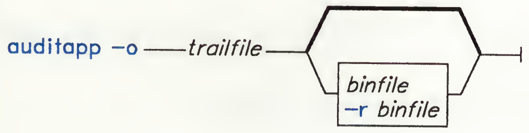
The discussion of accountability in *Managing the AIX Operating System.*

auditapp

Purpose

Adds an audit bin file to the end of the audit trail file.

Syntax



OL805474

Description

The **auditapp** command adds the audit records read from standard input to the audit trail file specified in the **/etc/security/audit/cmds** file. This command is part of the auditing system, which is fully discussed in *Managing the AIX Operating System.*

If you specify a bin file, (*binfile*), then **auditapp** reads from *binfile.*

If *trailfile* does not exist, **auditappend** creates the file.

This command is designed to be used by **auditbin** (a daemon) and should not be used on the command line. The **auditapp** command expects to find a complete bin (has both header and trailer portions). Entering **audit shutdown** completes the bin for processing by **auditapp**. Error conditions occur if the **auditapp** command is executed when bins are not properly completed with header and trailer portions.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-o	Specifies the the audit trail to which **auditapp** appends records. You must specify this flag.

-r	Recovers bin files before processing them. When you specify **-r** during a recovery procedure, **auditapp** will recover and process any unprocessed audit records. If you specify the **-r** flag, you must also specify *binfile*; however, you can specify *binfile* without the **-r** flag.

Files

etc/security/audit/cmds	Contains audit bin backend programs.

Related Information

The following commands: "**audit**" on page 67, "**auditbin**" on page 71, and "**auditselect**" on page 76.

The discussion of the audit trail in *Managing the AIX Operating System.*

auditbin

Purpose

Manages bins of audit information.

Syntax

```
auditbin ──┤
```

OL805475

Description

The **auditbin** command (a daemon) delivers ***bins*** of audit records to audit ***backends***. A bin is a file for storing audit information prior to processing. A backend is a program that sends its output, in this case the processed audit records, to a particular device or file. This device may then provide long-term storage. The default backend command is **auditapp** (see "**auditapp**" on page 69). This command is part of the auditing system, which is fully discussed in *Managing the AIX Operating System.*

Each audit backend is a command listed in **/etc/security/audit/cmds**. When **auditbin** receives a bin from the kernel, **auditbin** invokes each command in the **/etc/security/audit/cmds** file to process the bin. The **auditbin** command searches each command line for the keyword **$bin** and replaces it with the path name of the bin file.

If a backend command fails, **auditbin** stops processing the bins. It sends a message to **/dev/console** alerting the user of the problem and indicating that the command be terminated. The message repeats every 60 seconds until the command is terminated.

The **auditbin** command assures that each backend encounters each bin at least once. In the case of multiple commands, the **auditbin** command does not guarantee that each audit backend command will complete before the next one begins. Synchronization depends on the individual commands. Each backend command must wait for any duplicate command to complete.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Files

/etc/security/audit/cmds	Lists audit backend commands.
/etc/security/config	Specifies the audit configuration.

Related Information

The following commands: "**audit**" on page 67 and "**auditpr**" on page 73.

The **audit**, **auditevents**, **auditlog** and **auditproc** system calls in *AIX Operating System Technical Reference.*

The discussion of the audit trail in *Managing the AIX Operating System.*

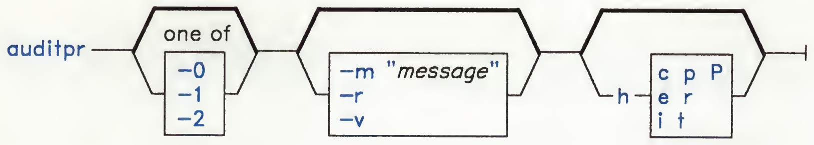
The following file format: **config** in *AIX Operating System Technical Reference.*

auditpr

Purpose

Displays audit trail files.

Syntax



A5AC5015

Description

The **auditpr** command reads kernel audit records from standard input and sends formatted records to standard output. This command is part of the auditing system, which is fully discussed in *Managing the AIX Operating System.*

By default **auditpr** searches the local **/etc/passwd** file to convert user and group IDs to names.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

The first series of flags, **-0**, **-1**, and **-2**, specify how often to print a title.

-0	Never print a title.
-1	This flag is the default. It specifies that a title be printed only once.
-2	This flag specifies that a title be printed before each record.
-m "*message*"	Displays *message* before each output record.
-r	Displays numeric user IDs.

-v Displays the tail of each audit record.

If the **-v** option is selected, **auditpr** prints the tail of each audit record in the format specified. The information in the tail is specific to the event that the record signifies. To print the tail of a record, **auditpr** searches the **auditpr** stanza of **/etc/security/audit/events** for an attribute name (audit event).

Note: The audit event is the attribute name. The attribute value has the form:

event = *path*[,"*arguments*"]

Where *path* specifies a command to be executed to print the tail of the record. Invoke this command as:

program arguments

The tail of the audit record is written to the program's standard input, and a formatted version is written to the program's standard output.

If an attribute is not found, **auditpr** will print as the tail the warning: `unknown event.`

-h *field* Displays header fields specified by *field*. The **-h** flag specifies the header fields to be printed. Field names and their widths are:

ID	Field	Width	Description
e	event	17	Audit event name.
c	command	17	Command name.
l	luid	6	User's login ID.
r	ruid	6	Process real user ID.
u	euid	6	Process effective user ID.
p	pid	6	Process ID.
P	ppid	6	Process ID of parent.
R	result	2	Result code of the action.
t	time	26	Time at which record was written.

The default header format is the combination **eclt**. The records that result from this default format appear as follows:

```
event      command luid   time
-----      ------- ----   ----
login      login   dick   Fri Feb 8, 1988 14:03:57
            . . .  tail portion, if requested  . . .
users      adduser jane   Fri Feb 8, 1988 14:04:33
            . . .  tail portion, if requested  . . .
```

For system calls, the tail portion consists of:

1. The arguments to the system call
2. A list of path names, each followed by two digits:
   - Nonzero - indicates that the path name is a symbolic link to the next path name. A zero (0) indicates a non-symbolic link file.
   - A return code by the kernel after trying to access the path name.

For items with the **printf** specification, the tail consists of the string of information specified in the quoted string that follows the specification.

File

/etc/security/audit/events Lists audit events.

Related Information

The following commands: "**audit**" on page 67 and "**auditselect**" on page 76.

The following system call: **audit** in *AIX Operating System Technical Reference.*

The following file formats: **attributes**, **events**, and **config** in *AIX Operating System Technical Reference.*

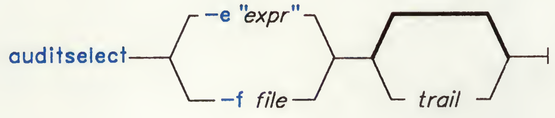
The discussions of hard copy labeling and the printer subsystem in *Managing the AIX Operating System.*

auditselect

Purpose

Selects audit records.

Syntax



OL805476

Description

The **auditselect** command reads audit records from standard input and writes records to standard output. This command is part of the auditing system, which is fully discussed in *Managing the AIX Operating System.*

If *trail* is specified, **auditselect** extracts records from audit *trail* and writes selected records to standard output.

The *file* is a file containing an expression.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-e "*expr*" Specifies an expression, *expr*, which consists of terms in the following form:

field relop value

These terms are defined as:

field One of the following:

- **event**
- **command**
- **login**
- **real**

- **effective**
- **pid**
- **ppid**
- **time**
- **prepend**

relop One of the following relational operation signs: ==(equal equal), !=(exclamation point equal), <(less than), >(greater than),. >=(greater than equal), or <=(less than equal).

value A quoted string if the event or command field was specified; a time in the format specified by the NLTIME environment variable; a date in the format specified by the NLDATE environment variable; or an integer if one of the following fields was specified: **pip, ppid, login, real**, and **effective**.

Combine these terms using the logical operators &&, (*and*) ||, (*or*) and ! (*not*). Use **()** (parentheses) to force the order of evaluation. Otherwise, normal precedence rules apply.

-f *file* Specifies a file containing an expression.

Related Information

The following commands: "**auditbin**" on page 71 and "**auditpr**" on page 73.

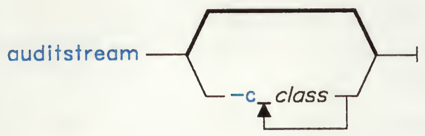
The **NLtmtime** subroutine in *AIX Operating System Technical Reference.*

auditstream

Purpose

Creates a channel for the reading of audit records.

Syntax



OL805477

Description

The **auditstream** command creates a channel to the audit device, **/dev/audit**. Audit records are read from **/dev/audit** by means of this channel and copied to standard output. The **auditstream** command can be used as the first command in an audit stream pipeline.

This command is part of the auditing system, which is fully discussed in *Managing the AIX Operating System.*

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Flags

-c*class*	Specifies audit classes as found in **etc/security/config**. Each audit record that belongs to an audit class specified by a **-c** option is read through the channel created by **auditstream**. If no audit classes are specified by a **-c** option in the **/etc/security/config** file, all currently enabled audit events are read through this channel.

Files

/etc/security/config	Specifies the audit configuration.
/dev/audit	The audit device.

Related Information

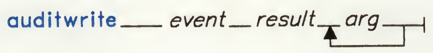
The discussion of the auditing subsystem in *Managing the AIX Operating System.*

auditwrite

Purpose

Generates an audit record at the command level.

Syntax



OL805478

Description

The **auditwrite** command combines an *event*, its *result*, and any *arguments* of supplied strings of data.

The *event* is the audit event to be audited (audit events can be found in the **/etc/security/audit/events** file), and the *result* is an indicator of the outcome of the *event*. The *arg* includes the audit information pertaining to the *event*.

This command is part of the auditing system, which is fully discussed in *Managing the AIX Operating System*. You must be a superuser to use this command.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Files

/etc/security/audit/events	Lists audit events.

Related Information

The following commands: "**audit**" on page 67, "**auditbin**" on page 71, "**auditpr**" on page 73, and "**auditselect**" on page 76.

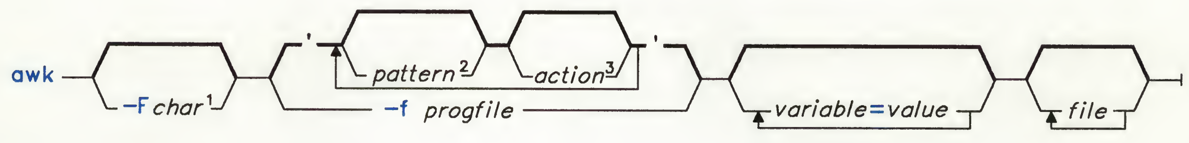
The discussion of auditing in *Managing the AIX Operating System*.

awk

Purpose

Finds lines in files matching specified patterns and performs specified actions on them.

Syntax



1 The default *char* is a tab.
2 The default pattern is every line.
3 The default action is to print the line.

OL805422

Description

The **awk** command is a more powerful pattern matching command than the **grep** command. It can perform limited processing on the input lines, instead of simply displaying lines that match. Some of the features of **awk** are:

- It can perform convenient numeric processing.
- It allows variables within actions.
- It allows general selection of patterns.
- It allows control flow in the actions.
- It does not require any compiling of programs.

For a detailed discussion of **awk**, see *AIX Operating System Programming Tools and Interfaces.*

The **awk** command, reads *file*s in the order stated on the command line. If you specify a file name as - (minus) or do not specify a file name, **awk** reads standard input.

The **awk** command searches its input line by line for *pattern*s. When it finds a match, it performs the associated *action* and writes the result to standard output. Enclose *pattern-action* statements on the command line in single quotation marks to protect them from interpretation by the shell.

The **awk** command first reads all pattern-action statements, then it reads a line of input and compares it to each pattern, performing the associated actions on each match. When it has compared all patterns to the input line, it reads the next line.

The **awk** command treats input lines as fields separated by spaces, tabs, or a field separator you set with the **FS** variable. Fields are referenced as **$1**, **$2**, and so on. **$0** refers to the entire line.

On the **awk** command line, you can assign *values* to variables as follows:

variable=value

Pattern-Matching Statements

Pattern-matching statements follow the form:

pattern { *action* }

If a *pattern* lacks a corresponding *action*, **awk** writes the entire line that contains the pattern to standard output. If an *action* lacks a corresponding *pattern*, it matches every line.

Actions

An action is a sequence of statements that follow C Language syntax. These statements can include:

statement	***format***
if	**if** (*conditional*) *statement* [**else** *statement*]
while	**while** (*conditional*) *statement*
for	**for** (*expression* ; *conditional* ; *expression*) *statement*
break	
continue	
{ *statement* . . . }	
(*assignment*)	*variable=expression*
print	**print** [*expression-list*] [>*expression*]
printf	**printf** *format*[, *expression-list*] [>*expression*]
next	
exit	

Statements can end with a semicolon, a new-line character , or the right brace enclosing the action.

If you do not supply an action, **awk** displays the whole line. Expressions can have string or numeric values and are built using the operators +, -, *, /, %, a blank for string concatenation, and the C operators ++, --, +=, -=, *=, /=, and %=.

Variables can be scalars, array elements (denoted x[i]) or fields. Variable names can consist of upper- and lower-case alphabetic letters, the underscore character, the digits (0-9), and SJIS characters.

Japanese Language Support Information

Variable names can also include kanji characters.

Variable names cannot begin with a digit. Variables are initialized to the null string. Array subscripts can be any string; they do not have to be numeric. This allows for a form of associative memory. String constants in expressions should be enclosed in double quotation marks.

There are several variables with special meaning to **awk**. They include:

FS	Input field separator (default is a blank). This separator character cannot be a 2-byte extended character.
NF	The number of fields in the current input line (record).
NR	The number of the current input line (record).
FILENAME	The name of the current input file.
OFS	The output field separator (default is a blank). This separator character cannot be a 2-byte extended character.
ORS	The output record separator (default is a new-line character). This separator character cannot be a 2-byte extended character.
OFMT	The output format for numbers (default `%.6g`).

Since the actions process fields, input white space is not preserved on the output.

The **printf** expression list formats like the **printf** subroutine (see *AIX Operating System Technical Reference*). It writes arguments to standard output, separated by the output field separator and terminated by the output record separator. You can redirect the output using the **print >** *file* or **printf >** *file* statements.

Note: You must enclose the file name in double quotes when redirecting output with the **awk** command.

You have two ways to designate a character other than white space to separate fields. You can use the **-F***c* flag on the **awk** command line, or you can start *progfile* with:

```
BEGIN { FS = c }
```

Either action changes the field separator to *c*.

There are several built-in functions that can be used in **awk** actions.

length [(*arg***)]**	Returns the length in characters of the whole line if there is no argument or the length of its argument taken as a string.
blength [(*arg***)]**	Returns the length in bytes of the whole line if there is no argument or the length of its argument taken as a string.
exp(*n***)**	Takes the exponential of its argument.

log(*n*)	Takes the base e logarithm of its argument.
sqrt(*n*)	Takes the square root of its argument.
int(*n*)	Takes the integer part of its argument.
substr(*s*,*m*,*n*)	Returns the substring *n* characters long of *s*, beginning at position *m*.
sprintf(*fmt*,*expr*,*expr*, . . .)	Formats the expressions according to the **printf** format string *fmt* and returns the resulting string.

Patterns

Patterns are arbitrary Boolean combinations of patterns and relational expressions (the !, ||, and && operators and parentheses for grouping). You must start and end patterns with slashes (/). You can use regular expressions like those allowed by the **egrep** command (see "**grep**" on page 501), including the following special characters:

+	One or more occurrences of the pattern.
?	Zero or one occurrences of the pattern.
|	Either of two statements.
( )	Grouping of expressions.

Isolated patterns in a pattern apply to the entire line. Patterns can occur in relational expressions. If two patterns are separated by a comma, the action is performed on all lines between an occurrence of the first pattern and the next occurrence of the second.

Regular expressions can contain extended characters with one exception: range constructs in character class specifications using square brackets cannot contain 2-byte extended characters. Individual instances of extended characters can appear within square brackets; however, 2-byte extended characters are treated as two separate 1-byte characters.

Japanese Language Support Information

Regular expressions can contain kanji characters. In that case, range constructs in character class specifications using square brackets can contain 2-byte kanji characters, which are treated as 2-byte characters.

Regular expressions can also occur in relational expressions. There are two types of relational expressions that you can use. One has the form:

expression matchop pattern

where *matchop* is either: ~ (for "contains") or !~ (for "does not contain"). The second has the form:

expression relop expression

where *relop* is any of the six C relational operators: <, >, <=, >=, ==, and !=. A conditional can be an arithmetic expression, a relational expression, or a Boolean combination of these.

You can use the special patterns **BEGIN** and **END** to capture control before the first and after the last input line is read, respectively. You can only use these patterns before the first and after the last line in *progfile*.

There are no explicit conversions between numbers and strings. To force an expression to be treated as a number, place a 0 at the beginning of the expression. However, note that only ASCII digits are treated as numeric. To force a regular expression to be treated as a string, append a null string ("").

Flags

-f *progfile*	Searches for the patterns and performs the actions found in the file *progfile*.
-F*char*	Uses *char* as the field separator character (by default a blank).

Examples

1. To display the lines of a file that are longer than 72 characters:

   ```
   awk  "length  >72"  chapter1
   ```

   This selects each line of the file chapter1 that is longer than 72 characters. **awk** then writes these lines to standard output because no *action* is specified.

2. To display all lines between the words start and stop:

   ```
   awk  "/start/,/stop/"  chapter1
   ```

3. To run an **awk** program (sum2.awk .) that processes a file (chapter1):

   ```
   awk  -f  sum2.awk  chapter1
   ```

   The following **awk** program computes the sum and average of the numbers in the second column of the input file:

   ```
       {
          sum += $2
       }

   END {
          print "Sum: ", sum;
          print "Average:", sum/NR;
       }
   ```

The first action adds the value of the second field of each line to the variable `sum`. **awk** initializes `sum` (and all variables) to zero before starting. The keyword **END** before the second action causes **awk** to perform that action after all of the input file has been read. The variable **NR**, which is used to calculate the average, is a special variable containing the number of records (lines) that have been read.

4. To print the names of the users who have the C shell as the initial shell:

```
awk  -F:  '/csh/{print  $1}'  /etc/passwd
```

Related Information

The following commands: "**lex**" on page 562, "**grep**" on page 501, and "**sed**" on page 887.

The **printf** subroutine in *AIX Operating System Technical Reference.*

The "Overview of International Character Support" in *Managing the AIX Operating System.*

The discussion of **awk** in *AIX Operating System Programming Tools and Interfaces.*

The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

back

Purpose

Plays backgammon.

Syntax

```
/usr/games/back ─┤
```

OL805186

Description

The **back** game provides you with a partner for backgammon. You select one of three skill levels: beginner, intermediate, or expert. You may also choose to roll your own dice during your turns, and you are asked if you want to move first.

The points are numbered such that:

- 0 is the bar for removed white pieces.
- 1 is white's extreme inner table.
- 24 is brown's extreme inner table.
- 25 is the bar for removed brown pieces.

For details on how to make your moves, enter `y` when **back** asks `Instructions` at the beginning of the game. When it first asks `Move?`, enter ? to see a list of choices other than entering a numerical move.

When the game is finished, **back** asks you if you want to save game information. A `y` response stores game data in the file **back.log** in your current directory.

The **back** game plays only the forward game, even at the expert level. It will object if you try to make too many moves in a turn, but not if you make too few. Doubling is not implemented.

To quit the game, press INTERRUPT (**Alt-Pause**).

Files

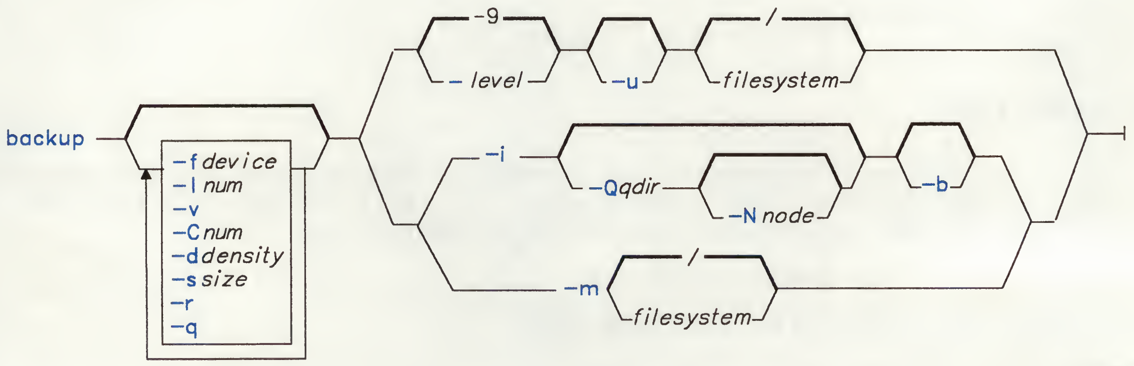
/usr/games/lib/backrules	Rules file.
/tmp/b*	Log temp file.
back.log	Log file.

backup

Purpose

Backs up files.

Syntax



OL805082

Description

The **backup** command copies files in **backup** format to a backup medium, such as a magnetic tape or diskette.

There are three ways to back up data:

- To back up specified files (***backup by name***) **-i**
- To back up an entire file system (***backup by file system or i-node***) *-level*
- To back up an entire minidisk (***backup by minidisk***) **-m**

To back up by name, use the **-i** flag. The **backup** command reads standard input for the names of the files to be backed up. You can specify files by using the **find** command to generate a list of path names and pipe the list into the **backup** command.

Backing up by name allows you to back up files to a backup medium on the local system or on a remote system.

When you specify **-Q**, **-Q** and **-N**, or when you use the **print -backup** command, the system writes a backup header to the backup medium. A header can contain the name of a qualifying directory and a target directory that subsequent **restore** commands can use to restore the files to the proper place. When a backup header is written if **-Q** is not specified, the system writes the path of the backup process's current directory to the header; if **-N** is not specified, the system writes the id of the node that requested the backup to the header.

To back up by file system (i-node), specify *-level* and *filesystem* to indicate the files you want to back up. You can use the level to back up either all files on the system (a ***full backup***) or only the files that have been modified since a specific full backup (an ***incremental backup***). The possible levels are 0-9. If you do not supply a level, the default level is 9. A level 0 backup includes all files on the file system. A level *n* backup includes all files modified since the last level *n*-1 backup. The levels, in conjunction with the **-u** flag, provide an easy way to maintain a hierarchy of incremental backups for each file system. For a discussion of backup strategy and the use of incremental backups, see *Managing the AIX Operating System.*

If you specify the name of a *filesystem*, it can be either the physical device name (the block or raw name) or the name of the directory on which the file system is normally mounted. When you specify a directory, **backup** reads **/etc/filesystems** for the physical device name. In this case, it also acquires values for other backup parameters from **/etc/filesystems**. If you do not specify a file system, the default is the root file system on the current minidisk.

To back up by minidisk, use the **-m** flag. This option copies an exact image of the entire minidisk. You can specify the file system name of the minidisk. The default is the root directory of the current minidisk. Because a backup by minidisk backs up an entire minidisk as an exact image, a large minidisk with a small or sparsely used file system may take longer and require more backup medium to back up this way, rather than by file system or by name.

When you do not specify a backup device, the **backup** command writes files to a default backup device. For backup by name, **backup -i**, the system writes to **/dev/rfd0** unless you specify a device with the **-f** flag. For a backup by file system (i-node), **backup** *-level*, or a backup by minidisk, **backup -m**, if **/etc/filesystems** contains a stanza that matches the name you specified and a stanza with a **backupdev** entry, then the system writes to the device specified by **backupdev**. Otherwise, the system writes to **/dev/rmt0** or the device specified with the **-f** flag.

The **backup** command recognizes a special syntax for the names of output files. If the argument is a range of names, such as **/dev/rfd0-3**, the **backup** command automatically goes from one drive in the range to the next. After exhausting all of the specified drives, it halts and requests that new volumes be mounted.

Notes:

1. During execution of remote backup operations, the file system is unmounted. The file system is remounted after the backup completes.
2. If you back up by either file system (i-node) or minidisk, the backup source and target must be on the local system. To back up to a remote system, back up by name with the **-i** flag. This flag allows users in a distributed services environment to back up files on a remote file system.
3. You should use the **-u** flag when you do an incremental backup to ensure that information regarding the last date, time, and level of each incremental backup is written to the file **/etc/budate**.
4. If the file system you are backing up is mounted and is not the root file system, **backup** unmounts the file system before it performs a file system (i-node) or minidisk backup and then remounts the file system before quitting. If the file systems you are backing up include the root file system, **backup** ensures that the other file systems are not in use. If one is, it warns you of this use and quits.

Warning: Be sure that the flags you specify match the backup medium. If the backup medium is not a disk or diskette, do not specify the **-l** flag. Similarly, if the backup medium is not a tape, do not specify the **-d** or **-s** flags. If you do specify flags that do not go with the medium, **backup** displays an appropriate error message and continues the backup.

Flags

-b Enables users to back up files in unattended mode (user input is not permitted) to a backup medium on a remote system. If any user input (such as `Please insert volume 2`) is required, the command ends in an error. This enables users to set up a shell file that backs up files at night or at other times when the user is unavailable.

-C*num* Specifies the number of blocks to write in a single output operation. If you do not specify *num*, **backup** uses a default value appropriate for the physical device selected. Larger values of *num* result in longer physical transfers to tape devices. The value of the **-C** flag is always ignored when **backup** writes to diskette. In this case, it always writes in clusters that occupy a complete track.

-d*density* Specifies the amount of data a system can write to a tape medium in bytes per inch. The default density is 700 bytes per inch.

Note: Tape drives vary in density capabilities. Use this flag with tape drives other than the IBM 6157 Streaming Tape Drive which has a density of 700.

-f*device*	Specifies the output device. Specify *device* as a file name (such as `/dev/rmt0`) to send output to the named device or specify **-** (minus) to send output to the standard output device. The **-** feature enables you to improve performance when backing up to streaming tape by piping the output of the **backup** command to the **dd** command (see example).
-i	Reads standard input for the names of files to back up.
-l*num*	Uses *num* as the limit of the total number of block to use on a diskette. The default value is the entire diskette (2400 blocks for 1.2M, 720 blocks for 360K diskette, and 2700 for rmt0 6157).
-m	Backs up the entire minidisk as an exact image.
-N *node*	Specifies the target node for subsequent **restore** commands. The *node* can be a node nickname or a node id (nicknames are translated to ids by **backup**). The **backup** command writes the id of node in the backup header. The default is the node id of the node where the **backup** command is running.
-q	Indicates that removable medium is ready to use. When you specify this flag, **backup** proceeds without prompting you to prepare the backup medium or waiting for you to press the **Enter** key to continue. Same as **-r** flag.
-Q *qdir*	Specifies the qualifying directory for subsequent **restore** commands. The **backup** command stores this name in the backup header. Then a subsequent **restore** command can use this information to place files with path names that are relative to a current directory in the qualifying directory. The *qdir* can be a relative or absolute directory. The default is the backup process's current working directory.
-r	Indicates that removable medium is ready to use. When you specify this flag, **backup** proceeds without prompting you to prepare the backup medium or waiting for you to press the **Enter** key to continue. Same as **-q** flag.
-s*length*	Specifies the *length* in feet of usable space on a tape medium. This is a combination of the physical length and the number of tracks on the tape. In the case of IBM RT Streaming Tape, you should multiply the physical length of the tape by 9 (the number of tracks) to determine the usable space available.
-u	Updates the time, date, and level of the backup in the **/etc/budate** file. This file provides the information needed for incremental backups.
-v	Reports on each phase of the backup as it is completed and gives regular progress reports during the longest phase.
-*level*	Specifies the backup *level* (0-9). The default *level* is 9.

Examples

1. To back up selected files:

   ```
   find  $HOME  -print  |  backup  -i  -v
   ```

   The -i flag tells the system to read from standard input the names of files to be backed up. The **find** command generates a list of files in the user's $HOME directory. This list is piped to the **backup** command as standard input. The -v displays a progress report as each file is copied. The files are backed up on the default backup device for the local system.

2. To back up an entire file system:

   ```
   backup  -0  -u  /
   ```

   The -0 level and the / file system tell the system to back up the entire root file system. The file system is backed up to the default device defined in the **backupdev** entry in **/etc/filesystems** if it exits. Otherwise, the files are backed up to **/dev/rfd0**. The **-u** tells the system to update the current backup level record in **/etc/budate**. Only the root file system is backed up, not mounted file systems.

3. To back up all files modified since the last level 0 backup:

   ```
   backup  -1  -u  /
   ```

4. To back up an entire minidisk:

   ```
   backup  -mf/dev/rmt1  /xyz
   ```

   This backs up the entire minidisk that contains the file system xyz. The -f tells the system to back up the minidisk to the streaming tape on /dev/rmt1 instead of the default device.

5. To back up files by name to the remote default device and specify the qualifying directory:

   ```
   find filelist -print | backup  -i -Q /tmp/darlene
   ```

   The system backs up the files in filelist and writes the qualifying directory /tmp/darlene to the header. Since a target node is not specified, the default node (the node where the **backup** command is running) is written to the header.

6. To back up files to a remote device and specify both the target node and the qualifying directory:

   ```
   find . -print | backup  -i -N darlene -Q /tmp/darlene
   ```

   This command backs up the current directory (.). The node nickname darlene is translated to a node id and written to the header with the qualifying directory /tmp/darlene. Note that when -N is specified, the -Q flag must also be present.

7. To improve performance on streaming tape, pipe the **backup** command to the **dd** command:

   ```
   backup -if- -C30 | dd of=/dev/rmt0 bs=30b
   ```

   The **backup** command backs up by name (`-i`), directs the output to the standard output device (`f-`), and specifies an output size as 30 blocks (`-C30`). The output is piped to **dd**. The **dd** command copies the files to an output file which is a streaming tape device (`of=/dev/rmt0`) and specifies a file size of 30 blocks (`bs=30b`). The file size in both commands should be the same. To restore these files, pipe the **dd** command to **restore**.

Files

/etc/filesystems	Read for default parameters.
/etc/budate	Log for most recent backup dates.
/dev/rfd0	Default backup device.
/dev/rhd0	Default file system.

Related Information

The following commands: "**find**" on page 422, "**dd**" on page 301, and "**restore**" on page 826.

The **budate** and **filesystems** files and the **tape** special file in *AIX Operating System Technical Reference.*

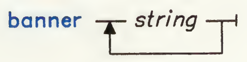
"Backing up Files and File Systems" in *Managing the AIX Operating System.*

banner

Purpose

Writes character strings in large letters to standard output.

Syntax



OL805080

Description

The **banner** command writes character *strings* to standard output in large letters. Each line in the output can be up to 10 uppercase or lowercase characters long. On output, all characters appear in uppercase, with the lowercase input characters appearing smaller than the uppercase input characters.

Examples

1. To display a banner at the work station:

   ```
   banner SMILE!
   ```

2. To display more than one word on a line, enclose the text in quotation marks:

   ```
   banner "Out to" Lunch
   ```

   This displays `Out to` on one line, and `Lunch` on the next.

3. To print a banner:

   ```
   banner We like Computers | print
   ```

Related Information

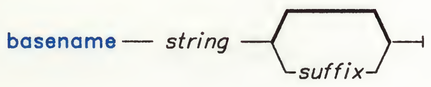
The following command: "**echo**" on page 369.

basename, dirname

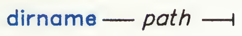
Purpose

Returns the base name of a string parameter.

Syntax



OL805085



OL805047

Description

The **basename** command reads the *string* specified on the command line, deletes any prefix that ends with a / (slash), as well as any specified *suffix*, if it is present, and writes the remaining base file name to standard output.

Note: A **basename** of / is null and is considered an error.

The **dirname** command writes to standard output all but the last part of the specified *path* name (all but the part following the last /).

The **basename** and **dirname** commands are generally used inside ***command substitutions*** within a shell procedure to specify an output file name that is some variation of a specified input file name. For more information, see "Command Substitution" on page 925.

Examples

1. To display the base name of a shell variable:

   ```
   basename  $WORKFILE
   ```

   This displays the base name of the value assigned to the shell variable `WORKFILE`. If `WORKFILE` is set to `/u/jim/program.c`, then `program.c` is displayed.

2. To construct a file name that is the same as another file name, except for its suffix:

   ```
   OFILE=`basename $1 .c`.o
   ```

   This assigns to `OFILE` the value of the first positional parameter (`$1`), but with its `.c` suffix changed to `.o`. If `$1` is `/u/jim/program.c`, then `OFILE` becomes `program.o`. Because `program.o` is only a base file name, it identifies a file in the current directory.

   The ` ` (grave accents) perform command substitution.

3. To construct the name of a file located in the same directory as another:

   ```
   AOUTFILE=`dirname $TEXTFILE`/a.out
   ```

   This sets the shell variable `AOUTFILE` to the name of an **a.out** file that is in the same directory as `TEXTFILE`. If `TEXTFILE` is `/u/fran/prog.c`, then the value of `dirname $TEXTFILE` is `/u/fran` and `AOUTFILE` becomes `/u/fran/a.out`.

Related Information

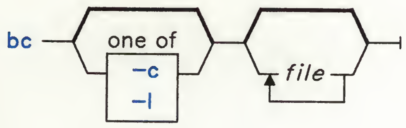
The following command: "**sh**" on page 913.

bc

Purpose

Provides an interpreter for arbitrary-precision arithmetic language.

Syntax



OL805081

Description

The **bc** command is an interactive process that provides unlimited precision arithmetic. It is a preprocessor for the **dc** command. **bc** invokes **dc** automatically, unless the **-c** (compile only) flag is specified. If the **-c** flag is specified, the output from **bc** goes to the standard output.

The **bc** command lets you specify an input and output base in decimal, octal, or hexadecimal (the default is decimal). The command also has a scaling provision for decimal point notation. The syntax for **bc** is similar to that of the C language.

The **bc** command takes input first from the specified *file*. When **bc** reaches the end of the input *file*, it reads standard input.

The following description of syntax for **bc** uses the following abbreviations: *L* means letters a-z; *E* means expressions; *S* means statements.

Names

Simple variables: *L*
Array elements: *L*[*E*]
The words **ibase**, **obase**, and **scale**.
Comments are enclosed in /* and */.

Other Operands

Arbitrarily long numbers with optional sign and decimal point.
(*E*)
sqrt (*E*)

length (*E*)	number of significant decimal digits
scale (*E*)	number of digits to the right of the decimal point

L (*E*, . . . ,*E*)

Operators

```
+ - * / % ^ (% is remainder; ^ is power)
+ + -- (prefix and postfix; apply to names)
= = <= >= != < >
= = + =- =* =/ =% =^
```

Statements

```
E
{ S; . . . ;S }
if (E) S
while ( E ) S
for (E;E;E) S
(null statement)
break
quit
```

Function Definitions

```
define L ( L, . . . ,L ) {

   auto L, . . . ,L
   S; . . . S
   return ( E )

}
```

Functions in -l Math Library

s(x)	sine
c(x)	cosine
e(x)	exponential
l(x)	log
a(x)	arctangent
j(n,x)	Bessel function

All function parameters are passed by value.

The value of a statement that is an expression is displayed unless the main operator is an assignment. A semicolon or new-line character separates statements. Assignments to **scale** controls the number of decimal places printed on output and maintained during multiplication, division, and exponentiation. Assignments to **ibase** or **obase** set the input and output number radix respectively.

The same letter may refer to an array, a function, and a simple variable simultaneously. All variables are global to the program. "Auto" variables are pushed down during function calls. When you use arrays as function parameters, or define them as automatic variables, empty square brackets must follow the array name.

All **for** statements must have all three E's.

The **quit** statement is interpreted when read, not when executed.

Flags

-c	Compiles *file*, but does not invoke **dc**.
-l	Includes a library of math functions.

Examples

1. To use **bc** as a calculator:

```
    You: bc
         1/4
 System: 0
    You: scale = 1  /* Keep 1 decimal place  */
         1/4
 System: 0.2
    You: scale = 3  /* Keep 3 decimal places */
         1/4
 System: 0.250
    You: 16+63/5
 System: 28.600
```

```
    You:  (16+63)/5
 System:  15.800
    You:  71/6
 System:  11.833
    You:  1/6
 System:  0.166
```

You may type the comments (enclosed in /* */), but they are provided only for your information. The **bc** command displays the value of each expression when you press the **Enter** key, except for assignments.

When you enter **bc** expressions directly from the keyboard, press **END OF FILE** **(Ctrl-D)** to end the **bc** session and return to the shell command line.

2. To convert numbers from one base to another:

```
    You:  bc
          obase = 16    /* Display numbers in Hexadecimal */
          ibase = 8     /* Input numbers in Octal         */
          12
 System:  A
    You:  123
 System:  53
    You:  123456
 System:  A72E
```

When you enter **bc** expressions directly from the keyboard, press **END OF FILE** **(Ctrl-D)** to end the **bc** session and return to the shell command line.

3. To write and run C-like programs:

```
    You:  bc -l prog.bc
          e(2)    /*  e squared   */
 System:  7.38905609893065022723
    You:  f(5)    /*  5 factorial */
 System:  120
    You:  f(10)   /* 10 factorial */
 System:  3628800
```

This interprets the **bc** program saved in `prog.bc`, then reads more **bc** statements from the work station keyboard. Starting **bc** with the **-l** flag makes the math library available. This example uses the **e** (exponential) function from the math library, and `f` is defined in the program file `prog.bc` as:

```
/* compute the factorial of n */

define f(n) {
   auto i, r;

   r = 1;
   for (i=2; i<=n; i++) r =* i;
   return (r);
}
```

The statement following a **for** or **while** statement must begin on the same line. When you enter **bc** expressions directly from the keyboard, press **END OF FILE (Ctrl-D)** to end the **bc** session and return to the shell command line.

4. To convert an infix expression to reverse polish notation (RPN):

```
You:    bc -c
        (a * b) % (3 + 4 * c)
System: lalb* 3 4lc*+%ps.
```

This compiles the **bc** infix-notation expression into one that the **dc** command can interpret. **dc** evaluates extended RPN expressions. In the compiled output, the **l** (ell) before each variable name is the **dc** subcommand to load the value of the variable onto the stack. The **p** displays the value on top of the stack, and the **s.** discards the top value by storing it in register . (dot). You can save the RPN expression in a file for **dc** to evaluate later by redirecting the standard output of this command. For more details, see "Redirection of Input and Output" on page 926. When you enter **bc** expressions directly from the keyboard, press END OF FILE (**Ctrl-D**) to end the **bc** session and return to the shell command line.

Files

/usr/lib/lib.b	Mathematical library.
/usr/bin/dc	Desk calculator proper.

Related Information

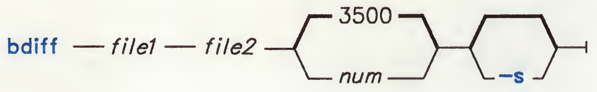
The following command: "**dc**" on page 295.

bdiff

Purpose

Uses **diff** to find differences in very large files.

Syntax



OL805083

Description

The **bdiff** command compares *file1* and *file2* and writes information about their differing lines to standard output. If either file name is **-** (minus), **bdiff** reads standard input. The **bdiff** command is used like **diff** to find lines that must be changed in two files to make them identical (see "**diff**" on page 320). Its primary purpose is to permit processing of files that are too large for **diff**.

The **bdiff** command ignores lines common to the beginning of both files, splits the remainder of each file into *num*-line segments, and calls **diff** to compare the corresponding segments. In some cases, the 3500 line default for *num* is too large for **diff**. If **diff** fails, specify a smaller value for *num* and try again.

The output of **bdiff** has the same format as that of **diff**. **bdiff** adjusts line numbers to account for the segmenting of the files. Note that because of the file segmenting, **bdiff** does not necessarily find the smallest possible set of file differences.

Flag

-s	Suppresses error messages from **bdiff**. (Note that the **-s** flag does not suppress error messages from **diff**).

Example

To display the differences between `chap1` and `chap1.bak`:

```
bdiff  chap1  chap1.bak
```

Files

/tmp/bd*	Temporary files.

Related Information

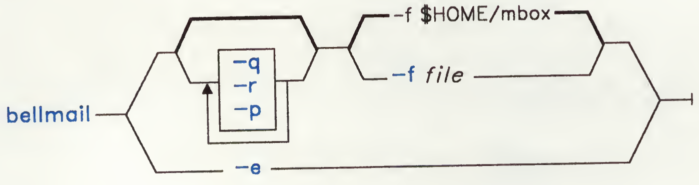
The following command: "**diff**" on page 320.

bellmail

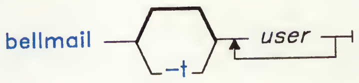
Purpose

Sends messages to system users and displays messages from system users.

Syntax



OL805347



OL805034

Description

The **bellmail** command with no flags writes to standard output, one message at a time, all stored mail addressed to the your login name. Following each message, **bellmail** prompts you with a ? (question mark). Press the **Enter** key to display the next mail message, or enter one of the subcommands that control the disposition of the message (see "Subcommands" on page 106).

When sending mail, you specify *users*, and then **bellmail** reads a message from standard input until you press **END OF FILE (Ctrl-D)** or enter a line containing only a . (period). It prefixes this message with the sender's name and the date and time of the message (its ***postmark***) and adds this message to the file **/usr/mail/***user* for each *user* specified on the command line.

The action of **bellmail** can be modified in two ways by manipulating **/usr/mail/***user*:

- The default permission assignment for "others" is "read-only." If you change this permission assignment to "read/write" or to "all permissions denied," the system preserves the file, even when it is empty, in order to maintain the desired permissions.
- You can edit the file to contain as its first line:

  `Forward to` *person*

  This causes all messages sent to *user* to be sent to *person* instead. The **Forward to** feature is especially useful for sending all of a person's mail to a particular machine in a network environment.

To specify a recipient on a remote system, prefix the system name and an exclamation mark (!) to *user*. See "**uucp**" on page 1144 for a detailed discussion of how to address remote systems.

Flags

-e	Does not display any messages. This flag causes **bellmail** to return an exit value of 0 if the user has mail, an exit value of 1 if he has no mail.
-f *file*	Saves mail in the named *file* instead of in the default mailfile, **$HOME/mbox**.
-p	Displays mail without prompting for a disposition code. This flag does not delete, copy, or forward any messages. (For disposition codes, see "Subcommands" on page 106).
-q	Causes **bellmail** to exit when you press INTERRUPT (**Alt-Pause**). Normally, pressing INTERRUPT (**Alt-Pause**) stops only the message being displayed. (In this case, the next message sometimes does not display until you enter the **p** subcommand.)
-r	Displays mail in first-in, first-out order.
-t	Prefixes each message with the names of all recipients of the mail. (Normally, only the individual recipient's name appears as addressee.)

Usually, *user* is a name recognized by the **login** command. It can also be the ASCII synonym that is automatically defined for any name that contains NLS code points. If the system does not recognize one or more of the specified *user*s or if **bellmail** is interrupted during input, **bellmail** saves messages in the file **$HOME/dead.letter** to allow for editing and resending.

Subcommands

The following subcommands control message disposition:

+	Displays the next mail message (the same as pressing the **Enter** key).
-	Displays the previous message.
d	Deletes the current message and displays the next message.
p	Displays the current message again.
s [*file*]	Saves the message in the named *file* instead of in the default mailfile, **$HOME/mbox**.
w [*file*]	Saves the message, without its postmark, in the specified *file* instead of in the default mailfile **$HOME/mbox**.
m *user*	Forwards the message to the named *user*.
q	Writes any mail not yet deleted to **/usr/mail/***user* and exits. Pressing END OF FILE **(Ctrl-D)** has the same effect.
x	Writes all mail unchanged to **/usr/mail/***user* and exits.
!*AIX-cmd*	Runs the specified AIX command.
*	Displays a subcommand summary.

Examples

1. To display your mail:

   ```
   bellmail
   ```

   After the most recent message is displayed, a ? (question mark) indicates that **bellmail** is waiting for one of the subcommands explained previously (+, -, **d**, **p**, etc.). Enter `help` or `*` (asterisk) to list the subcommands available.

2. To send mail to other users:

   ```
   bellmail tom rachel
   Don't forget the
   meeting tomorrow at 9:30.
   ```

   Ctrl-D

   In this example the system mails the message `Don't forget the meeting tomorrow at 9:30.` to the users `tom` and `rachel`. The **Ctrl-D** indicates the end of the message but it is not sent with the text.

3. To send a file to another user:

```
bellmail fran <proposal
```

This command sends the contents of the file `proposal` to `fran`. You can create `memo` with an editor, which allows you to correct your mistakes before sending the message. You can also use this form of the **bellmail** command to send someone a copy of a data file.

4. To retrieve a file that was sent to you:

```
bellmail
```

This command displays the messages mailed to you one at a time. You need to look at them because the file you want was actually added to **/usr/mail/***user* as a message. You may see several other messages before the file that was sent to you. If so, press the **Enter** key after the ? prompt until the desired file appears. If you go too far, enter the - (minus) subcommand to go back a message. After the ? immediately following the file, enter:

```
w mycopy
```

This command creates a file named `mycopy` in the current directory that contains the text mailed to you. Actually, you can save a copy of any message this way.

Files

/etc/passwd	To identify sender and locate *user*.
/usr/mail/*user*	Incoming mail for *user*.
$HOME/mbox	Saved mail.
$HOME/dead.letter	Unmailable text.
/tmp/ma*	Temporary file.
/usr/mail/*.lock	Lock for mail directory.

Related Information

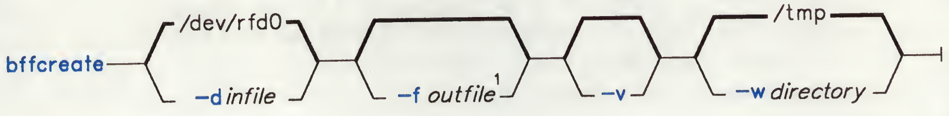
The following commands: "**login**" on page 584, "**uucp**" on page 1144, "**sendmail**" on page 897, and "**write**" on page 1225.

bffcreate

Purpose

Creates files in backup format for complete or subset programs in a code service environment.

Syntax



[1]See flag description for special requirements, restrictions, and defaults

AJ2FL137

Description

The **bffcreate** command creates one or more files in backup format to support install and update by client systems in a code service environment. Input files must also be in backup format. You must be a member of the system group or operating with superuser authority to run this command. This command is also run when you specify the **-b** flag in either an **installp** or **updatep** command.

This command creates one or more of the following:

- A file that contains the files from an **installp** distribution media
- One file per program subset that contains the program subset file from an **installp** distribution media
- A file that contains the files from an **updatep** distribution media
- A file that contains the files that do not follow **installp** or **updatep** conventions.

When this command runs, the contents of the distribution media are restored in a temporary working directory. Then a copy is created in backup format and put into either the **/usr/lpp.install** or the **/usr/lpp.update** directory for use in a code service environment. Program subset files are created automatically for any program subset on the distribution media.

Flags

-d *infile*	Specifies the name of the distribution media. If given, it must already exist. The default is **/dev/rfd0**.
-f *outfile*	Specifies the name of the backup format file. This flag is required for non-standard distribution media and **installp** distribution media that contain multiple program names. This flag is not allowed for distribution media that contain only one program name; in this case, the system names the output file in the form *programname.vv.rr* where *programname* is the name of the program, *vv* is the version, and *rr* is the release. This flag is optional for **updatep** distribution media. If not specified, the system creates a name in the form **updt.***yyddd.nnn* where *yyddd* is the Julian date (for example, 88032 is February 1, 1988) and *nnn* represents a number in sequence for files created that day.
-v	Writes the name of the backup format file to standard output (verbose mode).
-w *directory*	Specifies the directory where a temporary working directory can be created to contain the restored files. The default is **/tmp**. If specified, the directory must already exist.

Files

/usr/lpp.install	Directory that contains files in backup format for use in installing complete or subset programs across a network.
/usr/lpp.update	Directory that contains files in backup format for use in updating complete or subset programs across a network.

Related Information

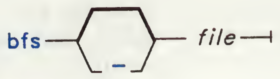
The following commands: "**installp**" on page 529 and "**updatep**" on page 1122.

bfs

Purpose

Scans files.

Syntax



OL805084

Description

The **bfs** command reads a *file* but does not do any processing of it, allowing you to scan but not edit it.

The **bfs** command is basically a read-only version of the **ed** command, except it can process much larger files and it has some additional subcommands. Input files can be up to 32K lines long, with up to 255 characters per line. **bfs** is usually more efficient than **ed** for scanning a file, because the file is not copied to a buffer. It is most useful for identifying sections of a large file where you can use the **csplit** command to divide it into more manageable pieces for editing.

If you enter the **P** subcommand, **bfs** prompts you with an * (asterisk). You can turn off prompting by entering a second **P**. **bfs** displays error messages when prompting is turned on.

Forward and Backward Searches

The **bfs** command supports all the address expressions described under "**ed**" on page 371. In addition, you can instruct **bfs** to search forward or backward through the file, with or without wrap-around. If you specify a forward search with wrap-around, **bfs** continues searching from the beginning of the file after it reaches the end of the file. If you specify a backward search with wrap-around, it continues searching backwards from the end of the file after it reaches the beginning. The symbols for specifying the four types of search are as follows:

/pattern/	Searches forward with wrap-around for the pattern.
?pattern?	Searches backward with wrap-around for the pattern.
>pattern>	Searches forward without wrap-around for the pattern.
<pattern<	Searches backward without wrap-around for the pattern.

The pattern matching routine of **bfs** differs somewhat from the one used by **ed** and includes additional features (see the **regcmp** subroutine in *AIX Operating System Technical Reference*). There is also a slight difference in ***mark names***: only lowercase letters **a** through **z** may be used, and all 26 marks are remembered.

Flags

-	Suppresses the display of file sizes. Normally, **bfs** displays the size in bytes of the file being scanned.

Subcommands

The **e**, **g**, **v**, **k**, **n**, **p**, **q**, **w**, **=** , **!** and null subcommands operate as explained under "**ed**" on page 371. Subcommands such as --, + + +-, + + + =, -12, and +4p are accepted. Note that `1,10p` and `1,10` both display the first ten lines. The **f** subcommand displays only the name of the file being scanned; there are no remembered file names. The **w** subcommand is independent of output diversion, truncation, or compression (see the **xo**, **xt**, and **xc** subcommands on page 111). ***Compressed output*** has strings of tabs and blanks reduced to one blank and blank lines suppressed.

The following additional subcommands are available:

xf *file*	Reads **bfs** subcommands from the *file*. When **bfs** reaches the end of file or receives an **INTERRUPT** signal or if an error occurs, **bfs** resumes scanning the file that contains the **xf** subcommand. These **xf** subcommands may be nested to a depth of 10.
xo [*file*]	Sends further output from the **p** and null subcommands to the named *file*, which is created with read and write permission granted to all users. If you do not specify a *file* parameter, **bfs** writes to standard output. Note that each redirection to a file creates the specified file, deleting an existing file if necessary.
:*label*	Positions a *label* in a subcommand file. The *label* is ended with a new-line character. Blanks between the : (colon) and the start of the *label* are ignored. This subcommand may be used to insert comments into a subcommand file, since labels need not be referenced.

[*addr1*[,*addr2*]]**xb**/*pattern*/*label*

Sets the current line to the line containing *pattern* and jumps to *label* in the current command file if *pattern* is matched within the designated range of lines. The jump fails under any of the following conditions:

- Either *addr1* or *addr2* is not between the first and last lines of the file.
- *addr2* is less than *addr1*.
- The pattern does not match at least one line in the specified range, including the first and last lines.

This subcommand is the only one that does not issue an error message on bad addresses, so it may be used to test whether addresses are bad before other subcommands are run. Note that the subcommand:

```
xb/^/label
```

is an ***unconditional jump***.

The **xb** subcommand is allowed only if it is read from some place other than a work station. If it is read from a pipe, only a ***downward jump*** is possible.

xt *number*	Truncates output from the **p** and null subcommands to *number* characters. The default *number* is 255.

xv[*digit*] [*value*] Assigns the specified *value* to the variable named *digit* (0 through 9). You can put one or more spaces between *digit* and *value*. For example:

```
xv5  100
xv6  1,100p
```

assigns the value 100 to the variable 5 and the value 1,100p to the variable 6.

To reference a variable, put a % (percent sign) in front of the variable name. Given the preceding assignments for variables 5 and 6, the following three subcommands:

```
1,%5p
1,%5
%6
```

each display the first 100 lines of a file.

To escape the special meaning of %, precede it with a \ (backslash). For example:

```
g/".*\%[cds]/p
```

matches and lists lines containing **printf** variables (%c, %d, or %s).

You can also use the **xv** subcommand to assign the first line of command output as the *value* of a variable. To do this, make the first character of *value* an ! (exclamation point), followed by the command name. For example:

```
xv5 !cat junk
```

stores the first line of the file junk in the variable 5.

To escape the special meaning of ! as the first character of *value*, precede it with a \ (backslash). For example:

```
xv7 \!date
```

stores the value `!date` in the variable 7.

xbz *label*	Tests the last saved exit value from a shell command and jumps to *label* in the current command file if the value is zero.
xbn *label*	Tests the last saved exit value from a shell command and jumps to *label* in the current command file if the value is not zero.
xc [*switch*]	Turns compressed output mode on or off. (Compressed output mode suppresses blank lines and replaces multiple blanks and tabs with a single space.) If *switch* is 1, output from the **p** and null subcommands is compressed; if *switch* is 0 it is not. If you do not specify *switch*, the current value of *switch* reverses. Initially, *switch* is set to 0.

Related Information

The following commands: "**csplit**" on page 252 and "**ed**" on page 371.

The **regcmp** subroutine in *AIX Operating System Technical Reference.*

biod

Purpose

Starts NFS asynchronous block I/O daemons.

Syntax

```
biod ── nservers ──┤
```

OL805479

Description

The **biod** command starts asynchronous block I/O daemons. This command is used on an NFS client to handle **read-ahead** and **write-behind** buffer cache. The *nservers* parameter specifies the number of asynchronous block I/O daemons started. Assign the number based on the load expected on the server. Four daemons can handle an average load.

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

File

/etc/rc.nfs

Related Information

The following command: "**nfsd**" on page 696.

biodd_cfg

Purpose

Configures the block I/O AIX device driver.

Syntax

```
biodd_cfg ──┤
```

AJ2FL143

Description

The **biodd_cfg** command configures the block I/O kernel device driver so it can access specific adapter cards. The adapter cards which can be accessed by the block I/O kernel device driver are adapters whose VRM device drivers are written to interface with the VRM block I/O device manager. These include the token adapter, the baseband adapter, and multiprotocol/dual port (MPDP) adapter.

Before you run this command, you must edit the **/etc/biodd** file to add the device names for the adapter cards you want to access. After the file has been edited, run the **biodd_cfg** command. This command must be run again after each IPL of the system for the block I/O kernel device driver to be configured to run with the devices listed in the file. To run this command automatically at each IPL, edit the **/etc/rc.include** file to uncomment the line:

```
# /etc/biodd_cfg
```

Japanese Language Support Information

If Japanese Language Support is installed on your system, this command is not available.

Files

/etc/biodd	Contains the device names of the adapters to be accessed by the block I/O kernel device driver.
/etc/rc.include	Contains startup routines.

Related Information

The discussion of the block I/O kernel device driver in *AIX Operating System Technical Reference.*

bj

Purpose

Plays blackjack.

Syntax

```
/usr/games/bj ─┤
```

OL805187

Description

The **bj** game plays the role of the dealer in blackjack. The following rules apply.

The bet is $2 every hand. If you draw a ***natural*** (blackjack), you win $3. If the dealer draws a natural, you lose $2. If you and the dealer both have naturals, you exchange no money (a ***push***). If the dealer has an ace showing, you can make an ***insurance*** bet on the chance that the dealer has a natural, winning $2 if the dealer has a natural and lose $1 if not. If you are dealt two cards of the same value, you can ***double***, that is, play two hands, each of which begins with one of these cards, betting $2 on each hand. If the value of your original hand is 10 or 11, you can ***double down***, that is, double the bet to $4 and receive exactly one more card in that hand.

Under normal play, you can draw a card (***hit***) as long as your cards total 21 or less. If the cards total more than 21, you ***bust*** and the dealer wins the bet. When you ***stand*** (decide not to hit), the dealer hits until he has a total of 17 or more. If the dealer busts, you win. If both you and the dealer stand, the one with the higher total wins. A tie is a push.

The **bj** command deals, keeps score, and asks the following questions at appropriate times: ? (Do you want a hit?) `Insurance? Double? Double down?`. To answer yes, press *y*; to answer no, press the **Enter** key.

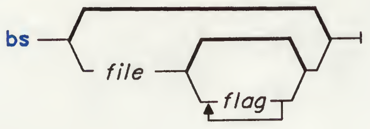
The dealer tells you whenever the deck is being shuffled and displays the ***action*** (total bet) and ***standing*** (total won or lost). To quit the game, press **INTERRUPT (Alt-Pause)**; **bj** displays the final action and standing and exits.

bs

Purpose

Compiles and interprets modest-sized programs.

Syntax



OL805167

Description

This compiler/interpreter provides interactive program development and debugging. To simplify program testing, it minimizes formal data declaration and file manipulation, allows line-at-a-time debugging, and provides trace and dump facilities and run-time error messages.

The optional command line parameter *file* specifies a file of program statements that the compiler reads before it reads from the standard input. By default, statements read from this file are compiled for later execution. Likewise, statements entered from the standard input are normally executed immediately (see the **compile** keyword on page 119 and the **execute** keyword on page 119). Unless the final operation is assignment, the result of an immediate expression statement is displayed.

Additional command line *flags* can be passed to the program using the built-in functions **arg** and **narg** (explained in more detail on page 123).

Program lines must conform to one of the following formats:

statement
label statement

The interpreter accepts labeled statements only when it is compiling statements. A *label* is a *name* immediately followed by a colon. A label and a variable can have the same name. If the last character of a line is a \ (backslash), the statement continues on the following physical line.

A statement consists of either an expression or a keyword followed by zero or more expressions.

Statement Syntax

break	Exits the innermost **for** or **while** loop.
clear	Clears the symbol table and removes compiled statements from memory. A **clear** is always executed immediately.
compile [*expr*]	Causes succeeding statements to be compiled (overrides the immediate execution default). The optional expression is evaluated and used as a file name for further input. In this latter case, the symbol table and memory are cleared first. **compile** is always executed immediately.
continue	Transfers control to the loop-continuation test of the current **for** or **while** loop.
dump [*name*]	Displays the name and current value of every global variable or, optionally, of the named variable. After an error or interrupt, **dump** displays the number of the last statement and (possibly) the user-function trace.
exit [*expr*]	Returns to the system level. The expression is returned as process status.
execute	Changes to immediate execution mode (pressing INTERRUPT [**Alt-Pause**] has the same effect). This statement does not cause stored statements to execute (see **run** on page 121).

for *name*=*expr expr statement*

for *name*=*expr expr*
  statement . . .
next

for *expr, expr, expr statement*

for *expr, expr, expr*
  statement . . .
next

Repeatedly performs, under the control of a named variable, a statement (first format) or a group of statements (second format). The variable takes on the value of the first expression, then is increased by one on each loop until it exceeds the value of the second expression. The third and fourth formats require three expressions separated by commas. The first of these is the initialization, the second is the test (true to continue), and the third is the loop-continuation action.

fun *f* ([*a*, . . .]) [*v*, . . .]
  statement . . .
nuf

Defines the function name (*f*), parameters (*a*), and local variables (*v*) for a user-written function. Up to 10 parameters and local variables are allowed. Such names cannot be arrays, nor can they be I/O associated. Function definitions may not be nested.

freturn	Signals the failure of a user-written function. Without interrogation, **freturn** returns zero. (See the unary interrogation operator **?** discussed on page 122.) With interrogation, **freturn** transfers to the interrogated expression, possibly bypassing intermediate function returns.
goto *name*	Passes control to the compiled statement with the matching label.
ibase *n*	Sets the input base to *n*. The only supported values for *n* are **8**, **10** (the default), and **16**. Hexadecimal values 10-15 are entered as alphabetic characters **a-f**. A leading digit is required when a hexadecimal number begins with an alphabetic character (for example, `f0a` must be entered as `0f0a`). **ibase** is always executed immediately.

if *expr statement*
 	
if *expr*
  statement . . .
[**else**
  statement . . .]
fi
	Performs a statement (first format) or group of statements (second format) if the expression evaluates to nonzero. The strings 0 and "" (null) evaluate as zero. In the second format, an optional **else** allows a group of statements to be performed when the first group is not. The only statement permitted on the same line with an **else** is an **if**; only other **fi**s can be on the same line with a **fi**. You can combine **else** and **if** into **elif**. Only a single **fi** is required to close an **if . . . elif . . . [else . . .]** sequence.

include *expr*	The expression must evaluate to the name of a file containing program statements. Such statements become part of the program being compiled. **include** statements may not be nested, and are always executed immediately.
obase *n*	Sets the output base to *n*. The only supported values for *n* are **8**, **10** (the default), and **16**. Hexadecimal values 10-15 are entered as alphabetic characters **a-f**. A leading digit is required when a hexadecimal number begins with an alphabetic character (that is, `f0a` must be entered as `0f0a`). Like **ibase**, **obase** is always executed immediately.
onintr *label* **onintr**	Provides program control of interrupts. In the first format, control passes to the label given, just as if a **goto** had been performed when **onintr** was executed. The effect of the **onintr** statement is cleared after each interrupt. In the second format, pressing **INTERRUPT (Alt-Pause)** ends **bs**.
return [*expr*]	Evaluates the expression and passes the result back as the value of a function call. If you do not provide an expression, the function returns zero.

run	Passes control to the first compiled statement. The random number generator is reset. If a file contains a **run** statement, it should be the last statement; **run** is always executed immediately.
stop	Stops execution of compiled statements and returns to immediate mode.
trace [*expr*]	Controls function tracing. If you do not provide an expression or if it evaluates to zero, tracing is turned off. Otherwise, a record of user-function calls/returns will be written. Each **return** decreases by one the **trace** expression value.
while *expr statement*	
while *expr* *statement* . . . **next**	**while** is similar to **for** except that only the conditional expression for loop continuation is given.
! *AIXcmd*	Runs an AIX command, then returns control to **bs**.
#comment	Inserts a comment line.

Expression Syntax

name	Specifies a variable or, when followed immediately by a colon, a label. Names are composed of a letter (uppercase or lowercase) optionally followed by letters and digits. Only the first six characters of a name are significant. Except for names declared locally in **fun** statements, all names are global. Names can take on numeric (double float) values or string values or be associated with input/output (see the built-in function **open** on page 125).
name([*expr*[, *expr*] . . . **)**	Calls function *name* and passes to it the parameters in parentheses. Except for built-in functions (listed in the following text), *name* must be defined in a **fun** statement. Function parameters are passed by value.
name[*expr*[, *expr*] . . .]	References either arrays or tables (see built-in function **table** on page 126). For arrays, each expression is truncated to an integer and used as a specifier for the name. The resulting array reference is syntactically identical to a name; `a[1,2]` is the same as `a[1][2]`. The truncated expressions must be values between 0 and 32767.
number	Represents a constant numerical value. This number can be expressed in integer, decimal, or scientific notation (it can contain digits, an optional decimal point, and an optional **e** followed by a possibly signed exponent).

string	Character string delimited by " " (double quotation marks). The \ (backslash) is an escape character that allows the double quotation mark (\"), new-line character (\n), carriage return (\r), backspace (\b), and tab (\t) characters to appear in a string. When not immediately followed by these special characters, \ stands for itself.
(*expr*)	Parentheses alter the normal order of evaluation.
(*expr*, *expr*[, *expr*] . . .) [*expr*]	The bracketed expression outside the parentheses functions as a subscript to the list of expressions within the parentheses. List elements are numbered from the left, starting at zero. The expression: `(False, True) [ a == b ]` has the value `True` if the comparison is true.
expr op expr	Except for the assignment, concatenation, and relational operators, both operands are converted to numeric form before the operator is applied.

Unary Operators

?*expr*	The interrogation operator (?) tests for the success of the expression rather than its value. It is useful for testing end of file, for testing the result of the **eval** built-in function, and for checking the return from user-written functions (see **freturn** on page 120). An interrogation ***trap*** (end of file, for example), causes an immediate transfer to the most recent interrogation, possibly skipping assignment statements or intervening function levels.
-*expr*	Negates the expression.
+ +*name*	Increases by one the value of the variable (or array reference).
--*name*	Decreases by one the value of the variable.
!*expr*	The logical negation of the expression.

Binary Operators (in increasing precedence)

=	The assignment operator. The left operand must be a name or an array element. It acquires the value of the right operand. Assignment binds right to left; all other operators bind left to right.
_	The concatenation operator (the underline character).
& ¦	Logical AND, logical OR. The result of: *expr & expr* is 1 (true) only if both of its parameters are nonzero (true); it is 0 (false) if one or both of its parameters are 0 (false).

	The result of: *expr* ¦ *expr* is 1 (true) if one or both of its expressions are nonzero (true); it is 0 (false) only if both of its expressions are 0 (false). Both operators treat a null string as a zero.
< <= > >= == !=	The relational operators (< less than, .<= less than or equal to, > greater than, >= greater than or equal to, == equal to, != not equal to) return 1 if the specified relation is True. They return 0 (false) otherwise. Relational operators at the same level extend as follows: **a>b>c** is the same as **a>b & b>c**. A string comparison is made if both operands are strings. The comparison is based on the collating sequence specified in the environment variable **NLCTAB**.
+ -	Addition and subtraction.
* / %	Multiplication, division, and remainder.
^	Exponentiation.

Functions Dealing With Arguments

arg(*i*)	Returns the value of the *i*-th actual argument at the current function call level. At level zero, **arg** returns the *i*-th command-line argument. For example, **arg(0)** returns `bs`.
narg( )	Returns the number of arguments passed. At level zero, it returns the command line argument count.

Mathematical Functions

abs(*x*)	Returns the absolute value of *x*.
atan(*x*)	Returns the arctangent of *x*.
ceil(*x*)	Returns the smallest integer not less than *x*.
cos(*x*)	Returns the cosine of *x*.
exp(*x*)	Returns e raised to the power *x*.
floor(*x*)	Returns the largest integer not greater than *x*.
log(*x*)	Returns the natural logarithm of *x*.
rand( )	Returns a uniformly distributed random number between zero and one.
sin(*x*)	Returns the sine of *x*.
sqrt(*x*)	Returns the square root of *x*.

String Functions

size(*s*)	Returns the size (length in characters) of *s*.
bsize(*s*)	Returns the size (length in bytes) of *s*.
format(*f*, *a*)	Returns the formatted value of *a*, *f* being a format specification string in the style of the **printf** subroutine. Use only the **%...f**, **%...e**, and **%...s** formats.
index(*x*, *y*)	Returns a number that is the first position in *x* containing a character that any of the characters in *y* matches. If there is no match, **index** yields zero. For 2-byte extended characters, the index functions returns the location of the first byte.
trans(*s*, *f*, *t*)	Translates characters in the source string *s* which match characters in *f* into characters having the same position in *t*. Source characters that do not appear in *f* are copied unchanged into the translated string. If string *f* is longer than *t*, source characters that match characters found in the excess portion of *f* do not appear in the translated string.
substr(*s*, *start*, *length*)	Returns the substring of *s* defined by *start*ing position in characters and *length* in characters.
match(*string*, *pattern*) **mstring(*n*)**	This function returns the number of characters in *string* that match *pattern*. The characters ., *, ? [,], ^ (when inside square brackets), \(and \) have the following special meanings (see "**ed**" on page 371 for a more detailed discussion of this special notation):

.	Matches any character except the new-line character.
*	Matches zero or more occurrences of the pattern element that it follows (for example, .* matches zero or more occurrences of any character except the new-line character).
$	Specifies the end of the line.
[.-.] [. . .]	Matches any one character in the specified range ([-.]) or list ([. . .]), including the first and last characters.
[^.-.] [^ . . .]	Matches any character except the new-line character and the remaining characters in the range or list. A circumflex (^) has this special meaning only when it immediately follows the left bracket.

[].-.]
[] . . .] Matches] or any character in the list. The right square bracket does not terminate such a list when it is the first character within it (after an initial ^, if any).

\(. . . \) Marks a substring and matches it exactly.

To succeed, a pattern must match from the beginning of the string. It also matches the longest possible string. Consider, for example:

```
match('a123ab123',".*\([a-z]\)") == 6
```

In this instance, `.*` matches `a123a` (the longest string that precedes a character in the range a-z); `\([a-z]\)` matches `b`, giving a total of six characters matched in the string. In an expression such as **[a-z]**, the minus means "through," according to the current collating sequence.

A collating sequence may define ***equivalence classes*** for use in character ranges. See the "Overview of International Character Support" in *Managing the AIX Operating System* for more information on collating sequences and equivalence classes.

Japanese Language Support Information

Note: Japanese Language Support does not define equivalence classes for use in character ranges. To avoid unpredictable results when using a range expression, use a ***character class expression*** rather than a standard range expression. For information about character class expressions, see "File Name Substitution" on page 4.

The **mstring** function returns the *n*th substring in the last call to **match** (*n* must be between 1 and 10 inclusive).

File-Handling Functions

open(*name*, *file*, *mode***)**
close(*name***)** The *name* parameter must be a legal variable name (passed as a string). For **open**, the *file* parameter may be:

- A **0**, **1**, or **2** for standard input, output, or error output, respectively
- A string representing a file name
- A string beginning with an !, representing a command to be run (via `sh -c`).

The *mode* flag must be either **r** (read), **w** (write), **W** (write without new-line character), or **a** (append). After a **close**, the *name* becomes an ordinary variable. The initial associations are:

```
open("get", 0, "r")
open("put", 1, "w")
open("puterr", 2, "w")
```

access(*p*, *m*) Performs the **access** system call. Parameter *p* is the path name of a file; *m* is a bit pattern representing the requested mode of access. This function returns a 0 if the request is permitted, -1 if it is denied. (See *AIX Operating System Technical Reference* for a more extensive discussion of this system call.)

ftype(*s*) Returns a single character indicating file type: **f** for regular file, **p** for FIFO (named pipe), **d** for directory, **b** for block special, or **c** for character special.

Table Functions

table(*name*, *size*) A table in **bs** is an associatively accessed, one-dimensional array. ***Subscripts*** (called keys) are strings (numbers are converted). The *name* parameter must be a **bs** variable name (passed as a string). The *size* parameter sets the minimum number of elements to be allocated. On table overflow, **bs** writes an error message.

item(*name*, *i*)
key( )

The **item** function accesses table elements sequentially (in normal use, there is an orderly progression of key values). Where the **item** function accesses values, the **key** function accesses the subscript of the previous **item** call. The *name* parameter should not be quoted. Since exact table sizes are not defined, the interrogation operator should be used to detect end-of-table; for example:

```
table("t",100)
.
.
.
#If word contains "party", the following expression
#adds one to the count of that word:
++t[word]
.
.
.
# To display the key/value pairs:
for i=0, ?(s=item(t, i)), ++i if key() put=key()_":"_s
```

iskey(*name*, *word*)
Tests whether the key *word* exists in the table *name* and returns one for true, zero for false.

Miscellaneous Functions

eval(*string*) The string parameter is evaluated as an expression. The function is handy for converting numeric strings to numbers. **eval** can also be used as a crude form of indirection, as in:

```
name = "xyz"
eval("++"_name)
```

which increments the variable xyz.

In addition, **eval** preceded by the interrogation operator permits you to control **bs** error conditions. For example:

```
?eval("open(\"X\",\"XXX\", \"r\")")
```

returns the value zero if there is no file named "XXX" (instead of halting your program). The following performs a **goto** to the label L: (if it exists):

```
label="L:"
if!(?eval("goto"_label))puterr="no label"
```

plot(*request*, *args*)
The **plot** function produces output on devices recognized by the **tplot** command. Some requests do not apply to all plotters. All requests except 0 and 12 are implemented by piping characters to **tplot**. The *requests* are as follows:

Call	***Function***
plot(0, *term***)**	Causes further **plot** output to be piped into **tplot** with a flag of **-T***term*.
plot(1)	Erases the plotter.
plot (2, *string***)**	Labels the current point with *string*.
plot(3, *x1*, *y1*, *x2*, *y2***)**	Draws the line between (x1, y1) and (x2, y2).
plot(4, *x*, *y*, *r***)**	Draws a circle with center (x, y) and radius *r*.
plot(5, *x1*, *y1*, *x2*, *y2*, *x3*, *y3***)**	Draws an arc (counterclockwise) with center (x1, y1) and endpoints (x2, y2) and (x3, y3).
plot(6)	Not implemented.
plot(7, *x*, *y***)**	Makes the current point at (x, y).
plot(8, *xy***)**	Draws a line from the current point to (x, y).
plot(9, *x*, *y***)**	Draws a point at (x, y).

plot(10, *string***)**	Sets the line mode to *string*.
plot(11, *x1*, *y1*, *x2*, *y2***)**	Makes (x1, y1) the lower left corner of the plotting area and (x2, y2) the upper right corner of the plotting area.
plot(12, *x1*, *y1*, *x2*, *y2***)**	Causes subsequent x (y) coordinates to be multiplied by x1 (y1) and then added to x2 (y2) before they are plotted. The initial scaling is **plot(12, 1.0, 1.0, 0.0, 0.0)**.

last() In immediate mode, **last** returns the most recently computed value.

Related Information

The following commands: "**ed**" on page 371, "**sh**" on page 913, and "**tplot**" on page 1079.

The **access** system call, the **printf** subroutine, and the **plot** file in *AIX Operating System Technical Reference.*

"Overview of International Character Support" in *Managing the AIX Operating System.*

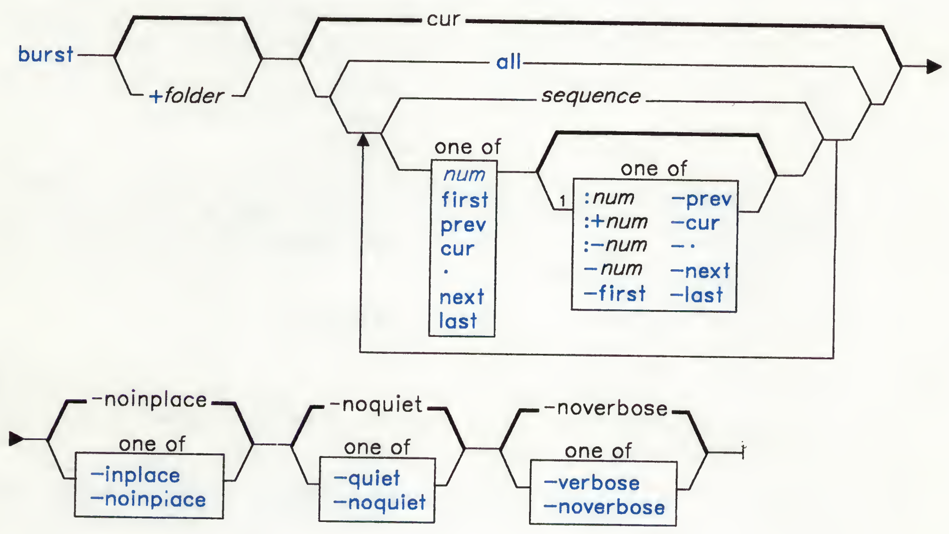
The discussion of Japanese Language Support in *Japanese Language Support User's Guide.*

burst

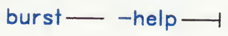
Purpose

Explodes digests into messages.

Syntax



AJ2FL220



AJ2FL160

¹ Do not put a blank between these items.

OL805308

Description

The **burst** command is used to explode digests, messages forwarded by the **forw** command, and blind carbon copies sent by the **forw** and **send** commands. **burst** is part of the MH (Message Handling) package and can be used with other MH and AIX commands.

The **burst** command cannot explode more than about 1,000 messages from a single message. **burst**, however, generally does not place a specific limit on the number of messages in a folder after bursting is complete.

The **burst** command uses encapsulation boundaries to determine where to separate the encapsulated messages. If an encapsulation boundary is located within a message, **burst** may split that message into two or more messages.

Flags

+*folder msgs* Specifies the messages that you want to burst. *msgs* can be several messages, a range of messages, or a single message. You can use the following message references when specifying *msgs*:

num	**first**	**prev**
cur	**.**	**next**
last	**all**	*sequence*

The default message is the current message in the current folder. If **-inplace** is also specified, the first message burst becomes the current message. Otherwise, the first message extracted from the first digest becomes the current message.

-help Displays help information for the command.

-inplace Replaces each digest by a table of contents for the digest, places the messages contained in each digest directly after the digest's table of contents, and renumbers all subsequent messages in the folder to make room for the messages in the exploded digest.

Warning: The **burst** command does not place text that appears after the last encapsulated message in a separate message. When you specify the **-inplace** flag, **burst** loses this trailing text. In digests, this text is usually an End-of-Digest string. However, if the sender appended remarks after the last encapsulated message, **burst** loses these remarks.

-noinplace Preserves each digest, does not produce a table of contents for each digest, and places the messages contained in each digest at the end of the folder. **burst** does not affect messages that are not part of digests. This flag is the default.

-noquiet Reports information about messages that are not in digest format. This flag is the default.

-noverbose	Does not report the general actions that **burst** performs while exploding the digests. This flag is the default.
-quiet	Does not report information about messages that are not in digest format.
-verbose	Reports the general actions that **burst** performs while exploding the digests.

Profile Entries

Current-Folder:	Sets your default current folder.
Msg-Protect:	Sets the protection level for your new message files.
Path:	Specifies your *user_mh_directory*.

Files

$HOME/.mh_profile	The MH user profile.

Related Information

Other MH commands: "**forw**" on page 438, "**inc**" on page 518, "**msh**" on page 677, "**packf**" on page 733, "**send**" on page 893, "**show**" on page 942.

The **mh-profile** file in *AIX Operating System Technical Reference.*

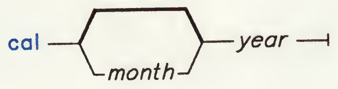
The "Overview of the Message Handling Package" in *Managing the AIX Operating System.*

cal

Purpose

Displays a calendar.

Syntax



OL805168

Description

The **cal** command writes to standard output a calendar for the specified year or month.

The *month* parameter names the month for which you want the calendar. It can be a number between 1 and 12 for January through December, respectively.

The *year* parameter names the year for which you want the calendar. Since **cal** can display a calendar for any year from 1 to 9999, enter the full *year* rather than just the last two digits.

Japanese Language Support Information

The name of the month is taken from the appropriate **NLLMONTH** string. The first two bytes of the **NLSDAY** environment variable string are used as the abbreviation of the day of the week.

Examples

1. To display a calendar for February 1984 at your work station:

   ```
   cal 2 1984
   ```

2. To print a calendar for 1984:

   ```
   cal 1984 ¦ print
   ```

3. To display a calendar for the year 84 A.D.:

   ```
   cal 84
   ```